目　　录

第0章　预备知识

本书叙述的内容不属于欧几里得的几何证明公理化范畴，而是属于欧几里得的几何构造，即由算法和复杂性分析所组成。欧几里得的几何构造满足算法的所有要求：无二义性、有穷性、确定性、能行性、输入、输出、正确性等。在欧几里得的几何构造中，限定了可允许使用的工具（直尺和圆规）及原始运算（圆规的一个脚置于一个给定点或一条直线上；作一个圆；直尺的边通过一个给定点；作一条直线）。但欧几里得原始运算并不能胜任所有的几何计算（比如角的三等分），这一点直到 19 世纪，阿贝尔、伽罗华等数学家才给出了证明。

在一个几何构造过程中，执行原始运算的总次数称为该过程的复杂性度量，这个概念对应于算法的时间复杂度。同样，还有对应于算法的空间复杂度的概念。这是欧几里得几何构造过程复杂性的定量测度。

称为计算几何的学科大致有下述几种：Forrest 等人依据样条函数处理曲线和曲面（实际上更接近于数值分析）；Minsky 和 Papert 写的一本名为《感知机》的书（副标题为"计算几何"），该书陈述用简单回路构成的网络实现模式识别的可能性（应属于人工神经网络）；计算机图形学是研究用计算机进行图形信息处理（包括表示、输入、输出、存储、显示、检索与变换等）和图形运算（如图的并、交运算）的一门学科，而不是算法分析；几何定理的机器证明，主要研究定理证明的探索方法及证明过程的推断，而不是几何本身。本书讨论的内容与上述计算几何学科所研究的内容不同，而是属于 Shamos 的文章(1975a)中命名的"计算几何"。为了不与上述"计算几何"的命名混淆，并突出 Shamos 的计算几何是研究几何问题的算法及复杂性的，故本书中将 Shamos 的计算几何称为 S 计算几何，而书名仍称为计算几何。

欧几里得货郎担问题、最小生成树问题、隐藏线（面）问题和线性规划问题等许多问题是 S 计算几何研究的基本问题。在 19 世纪的文献中，已经出现了对这些问题的算法研究，但对几何问题进行几何算法的系统研究还是近 20 多年的事情。

本书将通过对几何问题的研究，在以下各章节中给出 S 计算几何的观点、研究方法与几种重要的几何结构。S 计算几何的一个基本观点是，经典的几何对象的表征常常不适合于有效算法的设计。因此有必要建立一些概念及相关的性质，以适应于有效算法的设计。

本章将介绍有关算法与数据结构的一些知识、几何知识及计算模型。这些内容是以后章节所需要的。

0.1　算法与数据结构

0.1.1　算法

众所周知，算法是求解一个问题类的无二义性的有穷过程。这里的过程是指求解问题

执行的一步一步动作的集合，每一步动作只需要有限的存储单元和有限的操作时间。另外，如果详细说明一台典型的计算机以及与这种计算机通信的语言，那么凡用这种语言编写的，可以在给定的计算机上执行的过程便称为算法。随机存取机器(RAM)、图灵机等可以作为典型的计算机，拟 ALGOL 语言作为描述算法而非执行的语言。应该指出，算法不等于程序，因此描述算法的方式将是多种形式的，比如在拟 ALGOL 语言的描述中可以使用数学记号和自然语言。为了把算法转换成上机程序，还需要进行编程工作。

算法的复杂性包括算法的时间复杂性和算法的空间复杂性。为了说明复杂性的概念，先介绍问题规模的概念。用一个与问题相关的整数量来衡量问题的大小，该整数量表示输入数据量的尺度，称为问题的规模。比如，行列式的规模可以用其阶数 n 作为它的规模；图问题的规模可以用其边数或顶点数作为它的规模等等。

利用某算法处理一个问题规模为 n 的输入所需要的时间，称为该算法的时间复杂性。它显然是 n 的函数，记为 $T(n)$。

类似地可以定义算法的空间复杂性 $S(n)$。

以下主要讨论算法的时间复杂性。由于一般不需要知道精确的时间耗费，只要知道时间耗费的增长率大体在什么范围内即可，因此我们引入算法复杂性阶的概念。

如果对某一正常数 c，一个算法在时间 cn^2 内能处理规模为 n 的输入，则称此算法的时间复杂性是 $O(n^2)$，读作"n^2 阶"。一般定义如下：

如果存在正常数 c 和 n_0，使得当 $n \geqslant n_0$ 时有 $T(n) \leqslant cf(n)$，则称 $T(n)$ 是 $O(f(n))$，记作 $T(n)=O(f(n))$。此时，$f(n)$ 是 $T(n)$ 的增长率的一个上界。

如果 $T(n)=O(f(n))$ 和 $f(n)=O(T(n))$ 同时成立，则称 $T(n)$ 是 $\theta(f(n))$，记作 $T(n)=\theta(f(n))$。此时，$T(n)$ 和 $f(n)$ 的增长率是同阶的。

如果存在正常数 c，使得对无穷多个 n 关系式 $T(n) \geqslant cf(n)$ 成立，则称 $T(n)$ 是 $\Omega(f(n))$，记作 $T(n)=\Omega(f(n))$。此时，$f(n)$ 是 $T(n)$ 的增长率的一个下界。

如果 $T(n)=O(f(n))$ 和 $T(n)=\Omega(f(n))$ 同时成立，则有 $T(n)=\theta(f(n))$。

根据上述定义，两个函数如果同阶，那么它们可以相差一个常数因子，还可以相差比阶低的项，即函数中的低阶项并不影响它的阶数。这时，如要进一步分析 $T(n)$，就应考虑常数因子和低阶项对 $T(n)$ 的影响。

一般情况下，都是取一个简单形式的函数作为阶数的规范表示，如 n^2, n^3, n^6 等。实用中也是这样处理的。

一个算法的时间复杂性如果是 $O(2^n)$，则称此算法需要指数时间。而时间复杂性如果是 $O(n^k)$(k 为有理数)，则称此算法需要多项式时间。当 n 很大时，指数时间和多项式时间存在很大的差别。以多项式时间为限界的算法称为有效算法。有效算法的一个本质特征是可以按常规的方法在较短的时间内用一个确定的计算装置进行机械的计算。该计算装置的典型模型是图灵机和 RAM(random access machine)机。

算法的时间复杂性分为最坏情况的时间复杂性和平均情况的时间复杂性。对于给定规模为 n 的各种输入，某算法执行每条指令所需要的时间之和的最大值，称为该算法的最坏情况的时间复杂性；对于给定规模为 n 的各种输入，执行每条指令所需要的时间之和的平均值，称为平均情况的时间复杂性(或期望复杂性或平均特性)。由于规模为 n 的"输

入”出现的概率不同，所以有时要考虑加权平均特性。

为了求平均特性，必须对输入量的分布作某种假设。然而切合实际情况的假设，在数学上往往不易处理。因此，确定平均特性比确定最坏情况的时间复杂性难。

在最坏情况的时间复杂性和平均特性的定义中，都提到“指令”。由于不同的机器执行某一指令（如加法指令）所需要的时间可能不同，因此同一算法在不同机器上执行时所需的时间也可能不同。为了消除这一差距，人们引入一个理想的计算模型，用它代替具体的机器，以建立时空复杂性概念。这个理想的计算模型称为随机存取机器(RAM)，它只有一个累加器，而且不允许自行修改指令。

RAM 机器由一条只读输入带，一条只写输出带，一个程序存储器，一个存储器和指令计数器组成，如图 0-1 所示。

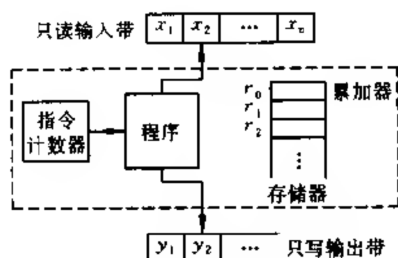


图 0-1 RAM 机器

输入带被划分为一系列方格，每个方格可以放一个整数。每当从输入带读出一个符号时，带头向右移动一格。输出带也被划分为一系列方格。执行一条写指令时，在输出带当前处于带头下的方格里打印一个整数，且带头右移一格，带头不能改写已印出的整数。

存储器由一系列寄存器 $r_0,r_1,r_n,\cdots$ 组成，每个寄存器能放下一个任意大小的整数。寄存器 r_i 的个数不受限定。

RAM 的程序不存放在存储器里，于是可假设程序不能修改自身。程序是带(也可不带)标号的指令序列，指令同实际的计算机一样，有算术指令、输入输出指令、间接寻址和转移指令等。用这些指令编写的程序称为 RAM 程序。RAM 的所有计算都在第一个寄存器 r_0(称为累加器)中进行。r_0 中也可放一个任意大小的整数。

RAM 程序的最坏情况时间复杂性和平均特性的定义与上面相同。

下面介绍最佳算法的概念。

在解决某一问题 P 的一类算法 $\mathscr{A}$ 中，需要操作次数最少的算法称为求解问题 P 算法类 $\mathscr{A}$ 的最佳算法。这里是用指定基本操作来确定算法类的。而算法 A 的基本操作是指 A 的关键操作，并且 A 的操作总数与基本操作次数成正比(当 n 增加时)。

形式上，设算法类 $\mathscr{A}$ 求解问题 P。对任一 $A\in\mathscr{A}$，其时间复杂性为 $T_A(n)$，定义

$$C_p(n)=\min_{A\in\mathscr{A}}\{T_A(n)\}$$

称为问题 P 关于 $\mathscr{A}$ 的计算复杂性。问题 P 的计算复杂性是 P 所固有的。由于 A 是 $\mathscr{A}$ 中任一算法，所以 $C_p(n)$ 难以估计。为此可以对问题 P 先求出在算法类 $\mathscr{A}$ 下 $C_p(n)$ 的下界(称为下界问题)，然后努力寻找达到这一计算复杂度的算法。

若能构造函数 $g(n)$，并可以证明，对任一 $A\in\mathscr{A}$，有 $T_A(n)\geqslant g(n)$ 成立，则 $g(n)$ 是 $C_p(n)$ 的一个下界。

设已找到求解 P 的算法 A，$A\in\mathscr{A}$，并分析 A 的复杂性 $T_A(n)$，有

$$C_p(n)\leqslant T_A(n)=f(n)$$

则 $f(n)$ 是 $C_p(n)$ 的一个上界。

设 $f(n)$，$g(n)$ 分别为 $C_p(n)$ 的上界和下界，如果 $f(n)=\theta(g(n))$，而且 $f(n)=T_A(n)$，则 A 是 $\mathscr{A}$ 中求解问题 P 的最佳算法（复杂性的阶不可能降低，但系数可能减少）。如果 $f(n)\neq\theta(g(n))$，则存在优于 A 的算法或者存在一个更低的下界。

设 A 是求解问题 P 的算法，那么可以用执行 A 时的"循环次数"作为算法时间复杂性的度量标准；在定义"最坏情况时间复杂性和平均特性"时，可考虑将"执行每条指令所需要的时间之和"作为度量标准；另外，"基本操作的次数"也可作为度量标准。前者使用方便，但估计粗糙；中者与机器的类型有关，需要引入一个抽象的计算模型，它对建立"最坏情况时间复杂性和平均特性"的概念有益；后者与机器、所采用的语言、实现方式等均无关，是较理想的度量标准。值得注意的是，我们只在同一种度量标准下比较某类算法中不同算法的优劣，而且在比较阶的同时，要特别注意以何种运算作为时间单位。如果时间单位不同，那么同一算法可能有不同的阶。

本书的主要目的是阐述几何问题的算法，以及对算法最坏情况复杂性的估计。

算法设计中的常用方法也适用于几何算法设计，比如分治法、贪心法、递归法、随机化方法和动态规划方法等。除了这些方法之外，依据几何问题的特征，将采用一些特殊的技巧，比如累接、修剪和搜索、几何变换、轨迹和扫描等方法。下面简要介绍平面扫描方法，其他方法将在后续章节中结合具体问题给予介绍。

给定平面上 n 条线段的集合 S，报告线段集合中的所有交点。考虑垂直直线 l，它将平面分割成两个半平面，即左半平面 L 和右半平面 R。显然，问题的解是 L 中的解和 R 中的解的并。当直线 l 穿过线段集合 S 时必与某些线段相交。另外，容易观察到，相交的两条线段和 l 的两个交点在 l 上是相邻的。因此，只要对 S 生成所有的垂直切割，就可以发现所有的交点。这就相当于把直线 l 从左向右扫过平面，故称平面扫描方法。当然，不可能连续地生成所有垂直切割的无限集合，但可以通过把平面划分为若干个垂直长条来实现，这些垂直长条由 S 中线段端点或交点确定。因此，只需从垂直长条的左边界跳到右边界，修改长条的次序，并测试 l 上新的相邻线段是否相交。l 从左向右扫过平面时，在称为"事件点"的特殊点处暂停，以便测试 S 中某些线段对是否相交。为此，需要两个基本结构：(1) 事件点进度表。S 中线段端点的 x 坐标的排序，它确定了扫描线 l 的暂停位置。执行平面扫描算法过程中，发现交点时便要修改事件点进度表。(2) 扫描线状态，它是扫描线 l 和 S 中线段的交点的一种描述。在每个事件点处修改扫描线状态。

0.1.2 数据结构

数据结构是组织信息的方式，它和算法一起使得问题可能得到有效的解。下面简要回顾有关的数据结构及其功能。

几何算法设计中遇到的对象是集合和序列。设 U 是某种几何对象的全集，u 是 U 中的任意元素，而 S 是 U 的子集。集合操作中的基本运算有：(1) MEMBER(u,S)：$u\in S$？(2) INSERT(u,S)：把 u 加入 S；(3) DELETE(u,S)：从 S 中删去 u。设 $\{S_1,S_2,\cdots,S_k\}$ 是一组两两不相交的集合，其上的运算有：(1) FIND(u)：若 $u\in S_i$，则报告 i；(2) UNION$(S_i,S_j;S_k)$：产生 S_i 和 S_j 的并，且记为 S_k。当 U 是全序时，有运算：(1) MIN(S)：报告 S 的最小元素；(2) SPLIT(u,S)：把 S 划分为 S_1 和 S_2，且 $S_1=\{v|v\in S$ 并且 $v\leqslant u\}$，$S_2=S$

$-S_1$;(3) CONCATENATE(S_1,S_2):设对于任意 $a\in S_1$ 和 $b\in S_2$,有 $a\leqslant b$,产生有序集 $S=S_1\cup S_2$。

对于有序集,依据支持的运算分类数据结构如下:字典,支持运算 MEMBER,INSERT,DELETE;优先队列,支持 MIN,INSERT,DELETE;可并队列,支持 INSERT,DELETE,SPLIT,CONCATENATE。通常以均衡二叉树来实现上述数据结构,此时,完成 INSERT 等运算的时间和存储在数据结构中的元素个数的对数成比例;存储和集合大小成比例。另外,上述数据结构还可以表示为一个线性的元素组(称为表),根据插入和删除的位置不同而分为队和堆栈,以 U_1 表示队或堆栈。记号"$\Rightarrow U_1$"表示插入 U_1,而"$U_1\Rightarrow$"表示从 U_1 中删去。

通过对无序集元素采用赋予"名字",且用字母次序的办法可以把无序集转变成有序集,这种情况的一种数据结构是可归并堆阵,它支持 INSERT,DELETE,FIND,UNION 等运算。仍以均衡树来实现该数据结构,并且耗费时间 $O(\log n)$完成 INSERT 等运算,其中 n 是存储在可归并堆阵中集合的大小。如无说明,本书中出现的 $\log n$ 均是以 2 为底的对数函数。

几何算法设计中广泛使用上述数据结构,此外,依据几何问题的特征,我们还采用线段树和双重连接边表等特殊的数据结构。

1. 线段树

线段树是一棵二叉树,记为 $T(a,b)$,并用 v 表示它的根,其中参数 a,b 是两个整数,它们表示一区间的始点和终点,记为 $B[v]=a$,$E[v]=b$。线段树 $T(a,b)$可以递归地构造如下:它包含一个根 v,根具有参数 a 和 b,且若 $b-a>1$,则它有左子树 $T(a,\lfloor(B[v]+E[v])/2\rfloor)$和右子树 $T(\lfloor(B[v]+E[v])/2\rfloor,b)$,其中"$\lfloor\ \rfloor$"表示低限。左、右子树的根分别为 LSON[$v$]和 RSON[$v$]。结点 v 所代表的区间由 $B[v]$和 $E[v]$确定,并满足关系式$[B[v],E[v]]\subseteq[a,b]$。图 0-2 表示线段树 $T(2,14)$。线段树的叶结点所代表的区间,称为基本区间,而其他结点表示$[B[v],E[v]]=[a,b]$中的某个子区间。容易将 $T(a,b)$建成均衡树(所有叶结点位于相邻的两极上)且具有深度 $\lceil\log(b-a)\rceil$,其中"$\lceil\ \rceil$"表示高限。

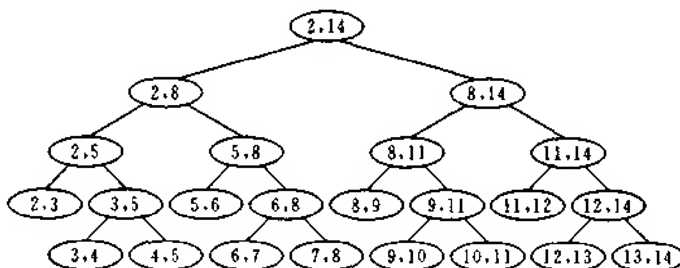


图 0-2 线段树 $T(2,14)$

线段树支持插入和删除运算。给定线段树 $T(a,b)$和任意区间$[c,d]$,下述过程将$[c,d]$插入线段树 $T(a,b)$。

```
procedure INSERT(c,d;v)
begin
  if  c≤B[v] ∧ E[v]≤d  then  把[c,d]分配给 v
  else  if  c< ⌊(B[v]+E[v])/2⌋  then  INSERT(c,d;LSON[v]);
        if  ⌊(B[v]+E[v])/2⌋ <d  then  INSERT(c,d;RSON[v])
end
```

INSERT$(c,d;\text{root}(T))$运算在 T 中沿下述路径进行：从根到结点(称为树叉)v^*的初始路径，称为 P_{IN}；从 v^*引出两条路径 P_L 和 P_R。或者把整个$[c,d]$分配给 v^*(此时，P_L 和 P_R 均为空)；或者整个分配给 P_L 结点的所有右子结点(这些右子结点不在 P_L 上)，以及 P_R 结点的所有左子结点(这些左子结点不在 P_R 上)，确定$[c,d]$的分配结点。如图 0-3 所示，其中双圆圈结点是分配结点。

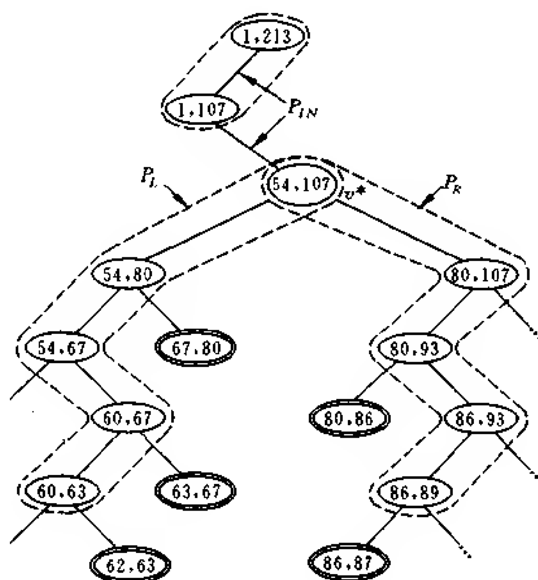


图 0-3　把区间[62,87]插入 $T(1,213)$

通常只需要知道分配给结点 v 的区间的集合的基数，用 $c[v]$表示该基数，然后如若将$[c,d]$再分配给 v，则有

$$c[v]\leftarrow c[v]+1$$

DELETE$(c,d;v)$运算表示如下：

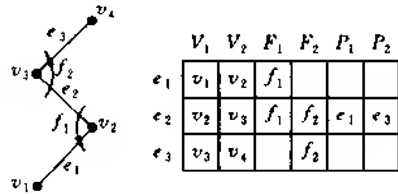
```
procedure DELETE(c,d;v)
begin
  if  c≤B[v] ∧ E[v]≤d  then  c[v]←c[v]−1
  else  if  c< ⌊(B[v]+E[v])/2⌋  then  DELETE(c,d;LSON[v]);
        if  ⌊(B[v]+E[v])/2⌋ <d  then  DELETE(c,d;RSON[v])
end
```

线段树是一种常用的数据结构。若要知道包含给定点 x 的区间个数，则对线段树 T 进行一次二叉查找（即从根到一个叶的一条路径的遍历）即可解决问题。

2. 双重连接边表

双重连接边表是另一种常用的数据结构，它适合于表示嵌入平面的平面图（连通）。给定平面图 $G=(V,E)$，G 的平面嵌入是将 V 中的顶点映射到平面中的一点，且 E 中的边映射到边的端点的两个像之间的一条简单曲线，所有简单曲线除端点外互不相交。可以取简单曲线为直线段。

设平面图 $G=(V,E)$ 的顶点集 $V=\{v_1,\cdots,v_n\}$，边集 $E=\{e_1,\cdots,e_m\}$。G 的双重连接边表如图 0-4 所示。每条边 e_i 给 4 个信息段 V_1，V_2，F_1 和 F_2 及两个指示字段 P_1 和 P_2，所以可以用相同名字的 6 个数组实现对应的数据结构，每个数组包含 m 个单元。各信息段的意义如下：V_1 与 V_2 分别为边 e_i 的起点和终点；F_1、F_2 分别表示位于有向边 e_i 的左侧面和右侧面的名字；指示字 $P_1(P_2)$ 指向一条边及端点，该边是边 $\overrightarrow{V_1V_2}$ 绕 $V_1(V_2)$ 按逆时针方向旋转后遇到的第 1 条边。



	V_1	V_2	F_1	F_2	P_1	P_2
e_1	v_1	v_2	f_1			
e_2	v_2	v_3	f_1	f_2	e_1	e_3
e_3	v_3	v_4		f_2		

图 0-4　双重连接边表的说明

从双重连接边表可以得到关联于给定顶点的边或者围绕一个给定面的边。假设 G 有 n 个顶点和 f 个面，另外，数组 $HV[1..n]$ 和 $HF[1..f]$ 分别是顶点和面表的首标，用 $O(n)$ 时间扫描数组 V_1 和 F_1 能填满这些数组。下述过程产生关联于 v_j 的边序列。

```
procedure VERTEX(j)
begin
  a←HV[j];
  a0←a;
  A[1]←a;
  i←2;
  if  V1[a]=j  then  a←P1[a]  else  a←P2[a];
  while  a≠a0  do
    A[i]←a;
    if  V1[a]=j  then  a←P1[a]  else  a←P2[a];
    i←i+1
end
```

VERTEX(j)的执行时间和关联于 v_j 的边数成比例。把上述过程中的 HV 和 V_1 分别换成 HF 和 F_1，便可得到围绕 f_j 的边序列。

边表是描述平面图 G 的另一种形式，对于每个顶点 $V_j\in V$，边表包含与 V_j 关联的边，且这些边按逆时针方向排列。显然耗费 $O(V)$ 时间可以把 G 的边表转换成双重连接边表。

0.2 相关的几何知识

0.2.1 基本定义

本书考虑的对象是欧几里得空间的点集，包含两个给定点的直线、直线上两定点确定的直线段、3个给定点确定的平面及由有序点列确定的多边形，等等。另外，假设有一个坐标系，使得每个点表示为相应维数的笛卡儿坐标的向量。我们还约定点集是有限可列举的。本节将简要回顾相关的几何知识，其细节请参看有关的资料。

用 E^d（或 R^d）表示 d 维欧几里得空间，即具有度量 $\left(\sum_{i=1}^{d}x_i^2\right)^{1/2}$ 的实数 $x_i(i=\overline{1,d})$ 的 d 元组 $(x_1,\cdots,x_d)$ 的空间。下面给出本书所涉及的基本对象的定义。

(1) **点** 用 p 表示一个点，E^d 中的点 p 定义为一个 d 元组 $(x_1,\cdots,x_d)$。点 p 也可解释为有 d 个分量的向量，此向量的起点为坐标原点，终点为点 p。

(2) **线，线性簇** 在 E^d 中给定两个不同的点 p_1 和 p_2，线性组合

$$\alpha p_1+(1-\alpha)p_2 \quad (\alpha \text{ 是实数，即 } \alpha\in\mathbf{R}) \tag{0-1}$$

是 E^d 中的一条线。如果给定 E^d 中 k 个线性独立的点 $p_1,\cdots,p_k(k\leqslant d)$，则线性组合

$$\alpha_1p_1+\alpha_2p_2+\cdots+\alpha_{k-1}p_{k-1}+(1-\alpha_1-\cdots-\alpha_{k-1})p_k \quad (\alpha_i\in\mathbf{R})$$

是 E^d 中的 $(k-1)$ 维的线性簇。

(3) **线段** 在 E^d 中给定两个不同的点 p_1 和 p_2，若在式(0-1)中加入条件 $0\leqslant\alpha\leqslant1$，则得到 p_1 和 p_2 的凸组合，它描述了连接两点 p_1 和 p_2 的直线段，并记为 $\overline{p_1p_2}$（无序对）。

(4) **凸集** 设 D 是 E^d 中的域，且 p_1 和 p_2 是 D 中的任意两点，如果线段 $\overline{p_1p_2}$ 完全包含在 D 中，则域 D 是凸的。

可以证明，两个凸域的交是一个凸域。

(5) **凸壳** E^d 中点的集合 S 的凸壳是 E^d 中包含 S 的最小凸域的边界。

(6) **多边形** 多边形定义为线段的有限集合，该集合中每条线段的端点恰好为两条边所共有，而且没有边的子集具有这个性质。线段为多边形的边，其端点是多边形的顶点，而且顶点数和边数相等。

若多边形的不相邻边对不相交，则称该多边形为简单多边形。简单多边形把平面划分为两个不相交的区域：内部区域（有界的）和外部区域（无界的）。一般情况下，多边形理解为边界和内部区域的并。

若简单多边形 P 的内部区域是凸集，则称 P 是凸的。如果 P 内存在点 q，使得对于 P 的所有点 p，线段 $\overline{qp}$ 完全位于 P 内，则此简单多边形是星形多边形。具有上述性质的 q 的轨迹称为 P 的核。

(7) **平面图** 若图 $G=(V,E)$ 能不交叉地嵌入平面，则 G 是平面图。平面图的直线平面嵌入确定平面的一个划分，称为平面剖分。设平面图的顶点数、边数和区域数分别为 v，e 和 f，则由欧拉公式有

$$v-e+f=2 \tag{0-2}$$

如果每个顶点的度数 $\geqslant3$，则 v，e 和 f 是两两成比例的。

(8) **三角剖分**　　若平面剖分的所有有界区域是三角形，则此平面剖分称为三角剖分。有限点集 S 的三角剖分是 S 上具有最大边数的平面图。或者说，由不相交的直线段来连接 S 的点得到 S 的三角剖分，以致每个三角形区域都在点集 S 凸壳的内部。

(9) **多面体**　　E^3 中，多面体定义为平面多边形的一个有限集，并且多边形的每条边和相邻的另一个多边形共有，而且没有多边形子集具有相同的性质。多边形的顶点和边是多面体的顶点和边；多边形是多面体的小面。

多面体中，如果没有不相邻的小面对共点，则此多面体称为简单多面体。简单多面体将空间划分为不相交的两部分（有界的内部域和无界的外部域）。一般说来，多面体是指边界和内部域的并。

无亏格的多面体的顶点、边和小面的数目 v、e 和 f 满足欧拉公式(0-2)。

0.2.2　线性变换群下的不变量

几何算法设计中，多数情况下是考虑欧几里得空间（二维和三维），并假设给定一个正交笛卡儿坐标系。但研究超出欧几里得空间限制的各种算法结果有效的范围是有意义的。也就是说，要描述保持给定算法有效空间的变换种类的特征。

可以把 E^d 中的点解释为它的坐标 d 个分量的向量 $(x_1, x_2, \cdots, x_d)$（或表示为 $\vec{x}$），而映射 $T: E^d \rightarrow E^d$ 解释为坐标点的移动（坐标系不变）。映射 T 中线性映射是最常用的一种，它可以描述如下：

$$x_i' = \sum_{j=1}^{d} a_{ji} x_j + C_i \quad (i = \overline{1, d}) \tag{0-3}$$

其中，点 p 的坐标为 $(x_1, x_2, \cdots, x_d)$，而 p 的像 p' 的坐标为 $(x_1', x_2', \cdots, x_d')$。或表示为

$$\vec{x}' = \vec{x} \boldsymbol{A} + \boldsymbol{C} \tag{0-4}$$

其中，$\boldsymbol{A} = (a_{ij})$ 是 $d \times d$ 矩阵，$\boldsymbol{C}$ 是指定的 d 个分量的向量，且所有向量是行向量。式(0-4)是仿射映射的一般形式，其中有两种特殊形式：$\boldsymbol{A} = \boldsymbol{I}$（单位矩阵）和 $\boldsymbol{C} = 0$。此时有

$$\vec{x}' = \vec{x} + \boldsymbol{C} \tag{0-5}$$

$$\vec{x}' = \vec{x} \boldsymbol{A} \tag{0-6}$$

式(0-5)是平移变换，在这种变换下每个点被移动 $\boldsymbol{C}$；式(0-6)将原点映射到自身，即原点是不动点。如果把 d 维向量 $(x_1, \cdots, x_d)$ 推广为 $d+1$ 维向量 $(x_1, \cdots, x_d, 1)$，则式(0-4)可写为

$$(\vec{x}', 1) = (\vec{x}, 1) \begin{pmatrix} \boldsymbol{A} & 0 \\ \boldsymbol{C} & 1 \end{pmatrix} \tag{0-7}$$

基于矩阵 $\boldsymbol{A}$ 的性质可以分类仿射映射。首先考虑 $\boldsymbol{A}$ 是任意非奇异矩阵，此时所有这种变换的集合形成一个群，称为仿射群，且仿射几何涉及到在此群中的变换之下保持不变的性质。仿射几何的基本不变量是关联的，即直线 l 上点 p 的成员资格。

其次考虑相似群，它是仿射群的一个子群，描述如下：

$$\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} = \lambda^2 \boldsymbol{I} \tag{0-8}$$

其中，λ 是实常数，上角标 T 表示转置。式(0-8)成立表明点之间的距离之比保持不变，所以角和正交性也不变。

再次要考虑的是正交群，它是在 $\boldsymbol{A}$ 的行列式 $|\boldsymbol{A}|=\pm 1$ 的特殊情况下得到的一种相似群。正交群的不变量是距离，所以面积、角、正交性也保持不变。保持距离不变的仿射变换是刚体运动，且形成了欧几里得几何的基础。

最后考虑描述 $d+1$ 维向量空间的特殊变换的式(0-7)，且除去变换矩阵最后一列的形式上的任何限制，得到关系式

$$\vec{\xi'} = \vec{\xi}\boldsymbol{B} \tag{0-9}$$

其中，$\vec{\xi'}$ 和 $\vec{\xi}$ 是 $d+1$ 维空间的欧几里得向量，$\boldsymbol{B}$ 是 $(d+1)\times(d+1)$ 矩阵，且设 $\boldsymbol{B}$ 为非奇异的。如果矩阵 $\boldsymbol{B}$ 的形式为

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{A} & 0 \\ \boldsymbol{C} & 1 \end{pmatrix}$$

则 E^d 的仿射群有新的解释：可以把任何二次曲线(椭圆、抛物线、双曲线)映射到圆，以致在群变换式(0-9)下所有二次曲线是等价的。此群称为射影群。

上述不同群之间的关系可以表示为图 0-5。

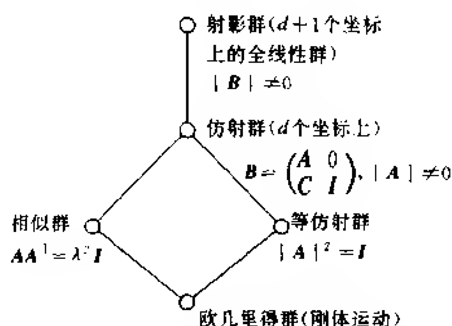


图 0-5　线性变换群上包含关系的说明

0.2.3　几何对偶性

当 $d=2$ 时，式(0-9)将点映射到点，线映射到线，即被映射的对象的维数保持不变。设 $|\boldsymbol{B}|\neq 0$，把式(0-9)写成

$$\vec{\eta} = \vec{\xi}\boldsymbol{B} \tag{0-10}$$

而且 $\vec{\xi}$ 表示的向量解释为过原点的一条线(即一个方向)，$\vec{\eta}$ 表示的向量解释为过原点的平面的法线。这样式(0-10)解释为 E^3 中一条线到一个平面的变换，称为对射变换(或对偶变换)，且能用相同矩阵 $\boldsymbol{B}$ 来描述把平面映射到线的对射。给定一个由矩阵 $\boldsymbol{B}$ 描述的线到平面的对射，寻找由矩阵 $\boldsymbol{D}$ 描述的线到平面的对射，使得对$(\boldsymbol{B},\boldsymbol{D})$保持关联，即若线 $\vec{\xi}$ 属于平面 $\vec{\eta}$，则线 $\vec{\eta}\boldsymbol{D}$ 属于平面 $\vec{\xi}\boldsymbol{B}$。$\vec{\xi}$ 属于平面 $\vec{\eta}$，当且仅当 $\vec{\xi}\cdot\vec{\eta}^{\mathrm{T}}=0$。乘积 $\boldsymbol{BD}$ 把线映

射到线，平面映射到平面。给定保持关联的一对($\boldsymbol{B}$,$(\boldsymbol{B}^{-1})^{\mathrm{T}}$)，要求乘积 $\boldsymbol{B}\cdot(\boldsymbol{B}^{-1})^{\mathrm{T}}$ 把每条线(平面)映射到它自身，即要求 $\boldsymbol{B}=k\boldsymbol{B}^{\mathrm{T}}$，其中 k 是某个常数。当 $k=1$ 时，有 $\boldsymbol{B}=\boldsymbol{B}^{\mathrm{T}}$，表明 $\boldsymbol{B}$ 是非奇异的 3×3 对称矩阵。

考虑如式(0-10)那样的中心射影到平面 $x_3=1$ 的对射的作用。此时点被映射到线，且线被映射到点。在式(0-10)中，$\vec{\xi}$ 是一个点，而 $\vec{\eta}$ 为一条线。若 3×3 矩阵 $\boldsymbol{B}$ 满足条件 $\boldsymbol{B}=\boldsymbol{B}^{\mathrm{T}}$，且 $\vec{x}=(x_1,x_2,x_3)$，则 $\vec{x}\boldsymbol{B}\vec{x}^{\mathrm{T}}=0$ 是平面中二次曲线的齐次方程，称为由 $\boldsymbol{B}$ 定义的二次曲线。现考虑指定的 $\vec{\xi}$(解释为平面中点的齐次表示)使得 $\vec{\xi}\boldsymbol{B}\vec{\xi}^{\mathrm{T}}=0$。由定义，点 $\vec{\xi}$ 位于 $\boldsymbol{B}$ 定义的二次曲线上。若称点 $\vec{\xi}$ 为极点，线 $\vec{\xi}\boldsymbol{B}\vec{x}^{\mathrm{T}}=0$ 为 $\vec{\xi}$ 的极线，则二次曲线上的极点属于它自己的极线。这个点到线和线到点的映射称为配极变换。它在几何算法设计中是有用的，这是由于处理点比处理线或平面更直观些。

如果选取 $\boldsymbol{B}$ 为平面 E^2 中定义单位圆的矩阵，此时

$$\boldsymbol{B}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&-1\end{pmatrix}$$

即把点 $p=(p_1,p_2,1)$ 映射到方程为

$$\vec{\xi}\boldsymbol{B}\vec{x}^{\mathrm{T}}=(p_1,p_2,1)\begin{pmatrix}1&0&0\\0&1&0\\0&0&-1\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=p_1x+p_2x_2-x_3=0$$

的线 l。从平面的原点到点 p 的距离是 $\sqrt{p_1^2+p_2^2}$，而从原点到 l 的距离是 $1\big/\sqrt{p_1^2+p_2^2}$。这表明在此特殊的配极变换中

$$d(p,o)\times d(\text{极线 }l,o)=\sqrt{p_1^2+p_2^2}\times\frac{1}{\sqrt{p_1^2+p_2^2}}=1$$

0.3 计算模型

在 0.1 节中介绍了随机存储机器 RAM，这是适用于现实计算机算法设计的一种计算模型。对于 RAM，我们能够指定可执行的原始运算和它们的耗费，也就是对每个原始运算支付一个指定的耗费。例如，若原始运算涉及到数的各个数位，则两个整数相加的耗费随操作数长度增加而增加；而在操作数具有固定长度的模型中，该耗费是常数。选取计算模型时，既要考虑贴近现实的计算又要考虑数学上是否易处理，为此必须抓住所用计算方法的主要特征，容许详尽的分析充分简单。我们要依据几何问题的特性选取适用于几何算法设计的模型。

S 计算几何中考虑的基本对象是点。虽然可以把点作为笛卡儿坐标系中的一个向量，但是几何算法的运行时间不受坐标系选取的较大影响，而是依赖于维数。因此可以在执行几何算法之前，假设在选取的笛卡儿坐标系中已给定点。

S计算几何中的问题大致可分为三类：

（1）提取子集　在一批给定的对象中提取满足某种性质的子集。这类问题的特征是不会产生新的对象，它的解完全由输入元素中的某些元素组成。比如，求点集的凸壳顶点的问题。

（2）计算　计算给定集合的某个几何参数的值。比如，计算点与线之间的距离，点对之间的距离等。

（3）判定　在"提取子集"和"计算"问题中都有判定问题。比如，在点集 S 中询问是否存在距离大于 d 的点对，回答"是"或"否"，其中 d 是已知常数。这是"计算"问题中的判定问题。另外，若提取子集问题 P 需要满足某个性质 Q 的给定集合 S 的子集，关联的判定问题 $D(P)$ 对于类型为"集合 S' 满足 Q?"的问题，要作"是"或"否"的回答。

如果省去实数近似表示中舍入问题的概念上的抽象，那么随机存储机器 RAM 可以用来作为几何算法设计所需要的计算模型。我们规定 RAM 的原始运算如下：(1) 算术运算；(2) 两个实数之间的比较；(3) 间接寻址及求根计算、三角函数、指数函数和对数函数等，并且执行每种运算一次耗费一个单位时间。

0.1 节中已经阐述了问题 P 的计算复杂性的下界，该下界对于评价算法的优劣是一个重要的标准。但一般来说，由于寻求下界是一个十分困难的任务，所以人们常采用问题变换（或称归约）的方法把一个问题与另一个问题联系起来。也就是说，给定两个问题 P_1 和 P_2，它们是关联的，并由下述步骤能求解问题 P_1：

步 1　把问题 P_1 的输入变换为问题 P_2 的适当的输入。

步 2　求解问题 P_2。

步 3　把问题 P_2 的输出变换为问题 P_1 的解。

如果能完成上述三个步骤，则称问题 P_1 可变换到问题 P_2。若用 $O(f(n))$ 时间能完成上述变换的步 1 和步 3，则称 P_1 是 $f(n)$ 可变换到 P_2，并记为 $P_1 \propto_{f(n)} P_2$。如果 $f(n)$ 是 n（问题 P_1 的规模）的多项式，则称 P_1 是多项式时间可变换到 P_2。

利用可变换性能求上、下界，有下述结论：

下界　若已知求解问题 P_1 需要 $T(n)$ 时间，且 $P_1 \propto_{f(n)} P_2$，则求解问题 P_2 至少需要 $T(n)-O(f(n))$ 时间。

上界　若求解问题 P_2 需要时间 $T(n)$，且 $P_1 \propto_{f(n)} P_2$，则求解 P_1 至多需要 $T(n)+O(f(n))$ 时间。

当 $f(n)=O(T(n))$ 时，即当变换时间不超出求解时间时，此转换成立。

如果 P_1 是计算问题或提取子集问题，与 P_1 关联的判定问题是 $D(P_1)$，那么可以直接判明 $D(P_1) \propto_{O(n)} P_1$。因为(1)若 P_1 是计算问题，则不需要输入变换（即步 1 没有用），只需用 $O(1)$ 时间把 P_1 的解和 $D(P_1)$ 提供的值比较；(2)若 P_1 是提取子集问题，则给 $D(P_1)$ 的输入 S' 为 P_1 的输入（步 1 也没有用），且用 $O(n)$ 时间检验 P_1 的解来证实它的基数和 S' 的基数相同。

因此，为了得到下界，我们仅考虑判定问题。

在 RAM 上执行求解判定问题的算法时，其过程可以描述为算术运算和比较运算的一个序列。序列中的比较运算起分支作用。因此，由 RAM 执行的计算可以视为有根树中

的一条路径，该树称为代数判定树，记为 T。树中的结点所代表的运算不是计算就是比较(分支)；叶结点表示计算终止，且包含可能的解。如果每次比较运算只产生两种结果，则判定树 T 是二叉树。显然，代数判定树的最大深度与 RAM 计算的最坏情况时间成比例，因此由代数判定树可以导出 RAM 计算所需时间的上界。

如果 T 中每个计算结点 v 由 d 次多项式组成，则称 T 是 d 阶的。$d=1$ 时，T 为线性判定树。线性判定树模型是建立问题的下界的有力工具。但若解决问题的算法中使用了 $d\geqslant 2$ 的函数，那么由线性判定树模型推得的下界就没有意义了。Steele，Yao 与 Ben-or 等人用实代数几何的概念解决了 $d\geqslant 2$ 的问题，下面简要介绍他们的结果。

设判定问题的参数为 $x_1,x_2,\cdots,x_n$，它的每个实例可作为 E^n 中的一点。判定问题确定点集 $W\subseteq E^n$，也就是说，判定问题回答"是"当且仅当 $(x_1,x_2,\cdots,x_n)\in W$，这时称判定树解决了 W 的成员资格问题。假设由问题的特性知道 W 的不相交的连通部分的个数 $\#(W)$。每个计算对应于 T 中的一条路径 v_1(根)，$v_2,\cdots,v_l$(叶子)，其中每个结点 v_j 关联一个 n 元函数 $f_{v_j}(x_1,\cdots,x_n)$，$j=\overline{1,l-1}$，使 $(x_1,\cdots,x_n)$ 满足约束 $f_{v_j}=0$ 或 $f_{v_j}\geqslant 0$ 或 $f_{v_j}>0$。

Dobkin 和 Lipton 解决了 $d=1$ 的情况(线性判定树或计算树模型)。设 $W\subseteq E^n$ 是给定判定问题的成员集，且 $\#(W)$ 是其不相交的连通部分的个数，T 是测试 W 中成员资格的算法 A 的二叉线性判定树。T 的叶子关联 E^n 的一个域，且当关联于叶子 l_j 的域 $D_j\subseteq W$ 时，接受叶子 l_j；否则拒绝 l_j。Dobkin 和 Lipton 的结论是：解 $W\subseteq E^n$ 中成员问题的任何线性判定树算法至少有深度 $\log\#(W)$，其中 $\#(W)$ 是 W 的不相交连通部分的个数。这个结论依赖于性质：关联于树的一个叶子的 E^n 域是凸的。而当与 v 关联的函数的最大次数 $\geqslant 2$ 时，该性质不成立，因为在这种情况下关联于判定树叶子的域可包含 n 个 W 的不相交部分。如果依据叶子深度能限定赋予叶子的组成部分的个数，则也可以限定 T 的深度。Steele，Yao 与 Ben-or 等人解决了这个问题，得到下面的结论：

设 W 是笛卡儿空间 E^n 中的集合，T 是解 W 中成员问题的阶 $d(\geqslant 2)$ 的代数判定树。若 T 的深度是 h，且 $\#(W)$ 是 W 的不相交的连通部分的个数，则 $h=\Omega(\log\#(W)-n)$。这个结果是讨论下界的基础。

第 1 章　几何查找(检索)

几何查找或者称几何检索是指在属性相同的一批几何对象(比如点、直线段、圆、多边形、多面体等)中定位某个指定的几何对象，或者在某个特定的域中寻找该域所包含的具有某种属性的所有几何对象。换句话说，几何查找包括两类主要的问题：(1) 几何体定位问题；(2) 范围查找问题。本章主要介绍求解这两类问题的算法。

一批几何对象所对应的数据称为文件，而指定的几何对象所对应的数据称为样本。几何查找就是在文件中查找样本。这里的查找常常不是将样本与文件中的项目匹配，而是在相关的文件中定位样本。如果在文件中仅执行一次查找，那么就没有必要对文件进行预处理，否则，就有必要预处理文件。所谓预处理文件是指将一批几何对象所对应的数据按一定的结构存放。在分析几何查找复杂性时，预处理所需要的时间和存储空间不能忽略。一般来说，几何查找的耗费包括询问时间(回答一次询问所要的时间)、存储(数据结构占用的内存)、预处理时间(组织数据或某种结构的时间)、修改时间(指定几何对象所对应的数据插入数据结构或从数据结构中删去所需要的时间)。下面用一个例子来说明询问时间、存储和预处理时间之间的各种折衷办法。

给定平面上点集 $S=\{p_1,p_2,\cdots,p_n\}$ 及边平行于坐标轴的矩形 R，矩形 R 的 4 个顶点及坐标分别为 $C(a,c)$，$B(a,d)$，$A(b,d)$，$D(b,c)$，问 S 中有多少个点位于 R 内？也就是要确定有多少个点 (x,y) 满足 $a\leqslant x\leqslant b$，$c\leqslant y\leqslant d$？这是范围查找问题。

解决这个问题的一种方法是逐点检查 S 中的点 $p_i(x_i,y_i)$ 是否满足不等式 $a\leqslant x_i\leqslant b$，$c\leqslant y_i\leqslant d$。假设检查一个点耗费的时间为一个单位时间，那么这种方法需要线性时间。另外，n 个点有 $2n$ 个坐标，假设存储一个坐标耗费一个单位空间，则该方法的空间复杂性是 $O(n)$。这种方法不需要预处理。如果回答 S 中一个点是否在 R 中所用的时间为一个单位时间，那么询问时间(回答 S 中哪些点在 R 中所耗费的时间)为 $O(n)$。

为了介绍解决这个问题的另一种方法，先引进向量优势的概念。点(向量)$p(x,y)$优于 $p'(x',y')$ 当且仅当对于坐标 x 和 y，$x\geqslant x'$ 并且 $y\geqslant y'$。用 $Q(p(x,y))$ 表示满足 $x'\leqslant x$ 并且 $y'\leqslant y$ 的点 $p(x',y')$ 的数目。这样，落入矩形 R(4 个顶点分别为 A,B,C,D)的点数

$$N(A\ B\ C\ D)=Q(A)-Q(B)-Q(D)+Q(C)$$

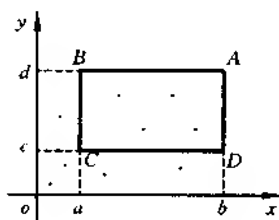


图 1-1　范围查找相当于 4 个优势询问

如图 1-1 所示。这表明，范围查找问题可以转化为 4 个点优势询问问题。从 S 中的点向 x，y 轴作垂线并延长垂线，这些垂线将形成 $(n+1)^2$ 个矩形的网格。对于任意给定矩形中的所有点 p，$Q(p)$ 是常量。这表明优势查找恰是决定给定点所在的矩形网格的哪一个区域的问题。为了回答这个问题，先按两个坐标分类点(或者称按两个坐标对点排序)，然后执行两次对分查找，寻找哪一个矩形包含此点。因此询问时间为 $O(\log n)$。但这种方法需要 $O(n^2)$ 空间，因为有 $O(n^2)$ 个矩形。对任一矩形，

用 $O(n)$ 时间计算矩形的优势数，所以预处理时间为 $O(n^3)$。后来，Bentley 和 Shamos 将预处理时间减少到 $O(n^2)$。他们还提出了另一种方法，使得询问时间为 $O(\log^2 n)$，预处理时间为 $O(n\log n)$，空间减少到 $O(n\log n)$。

1.1 点定位问题

点定位问题(即点 p 位于区域 R 中)与点包含问题(即区域 R 内包含点 p)的含义是相同的。这类问题的求解主要依赖于空间的划分。在平面情况下，考虑直线段构成的平面划分(又称平面剖分)，并且这种划分形成的子区域是连通的。现在我们考虑下列问题。

1.1.1 点 q 是否在多边形 P 内

问题 1-1 给定简单多边形 P 及点 q，如何确定 P 是否包含 q 或者 q 是否在 P 内？

设简单多边形 P 的顶点序列为 $p_1, p_2, \cdots, p_n$，求解问题 1-1 的一种方法是过点 q 作水平射线 l，如果 l 与 P 的边界不相交，则 q 在 P 的外部。否则，l 和 P 的边界相交，计数交点数并依据交点数的奇偶性可以判定点 q 是否在 P 的内部，具体地说，交点数为奇(偶)数时，点 q 在 P 的内(外)部。由于这里的多边形不一定是凸的，所以上述的判断方法对某些特殊情况是不适合的，要注意区分这些特殊情况，这些情况如图 1-2 所示。

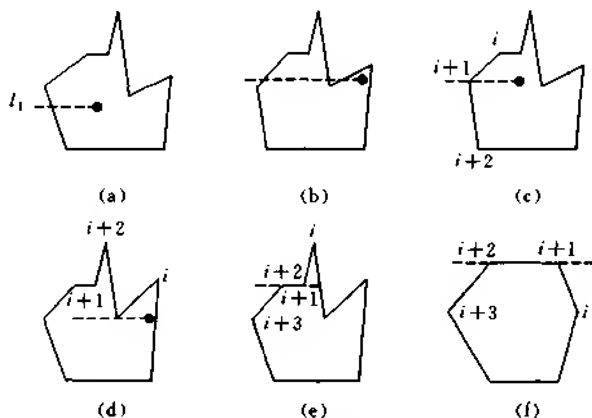


图 1-2 几种特殊情况

图 1-2 中(a)是正常情况，有一个交点。(b)有三个交点。这两种情况均表明点在多边形内部。(c)应该看成有一个交点。(d)看成没有交点。(e)和(f)有多个交点。我们想办法把这些情况区分开，为此引入两个函数 intersect(l_1,l_2)和 online(l,p)。仅当 l_1 和 l_2 相交而且不交在端点上时，intersect(l_1,l_2)才为真。当 p 在 l 上时 online(l,p)才为真。如果 $p[i]$ 和 $p[i+2]$ 在 l 的两侧，则是图 1-2(c)的情况。如果 $p[i]$ 和 $p[i+2]$ 在 l 的同侧，则是图 1-2(d)的情况。如果 $p[i]$ 和 $p[i+3]$ 在 l 的两侧(或同侧)，则是图 1-2(e)(或(f))的情况。

下面是计算 online 函数和判断一个点是否在多边形内部的程序(设多边形有 n 个

顶点)。

```
var p:array[1..n+1] of point;
function online (l: line; p:point): boolean;
var dx,dy,dx1,dy1: real;
begin
        dx←l.p2.x−l.p1.x
        dy←l.p2.y−l.p1.y
        dx1←p.x−l.p1.x
        dy1←p.y−l.p1.y
        online ←(dx * dy1−dy * dx1=0) and
                (dx1 * (dx1−dx)<0 or dy1 * (dy1−dy)<0)
end;
function inside (q:point): boolean
var c,i:integer
    l1,l2:line
begin
        c←0
     l1.p1←q
     l1.p2←q
   l1.p2.x←maxint
     for i=1 to n do
       begin
         l2.p1←p[i]
         l2.p2←p[i+1]
         if  intersect(l1,l2)  or  online(l1,p[i+1])
             and [not online (l1,p[i+2]) and not
             same 1 (l1,p[i],p[i+2]) or online (l1,p[i+2])
             and not same 1 (l1,p[i],p[i+3])]
           then c←c+1
       end
       inside←(c mod 2<>0)
end
```

其中函数 same 1 为

$$\text{same }1=(dx*dy_1-dy*dx_1)*(dx*dy_2-dy*dx_2)\geqslant 0$$

由于 P 有 n 条边,要检查每条边是否与 l 相交,所以该算法的时间复杂性为 $O(n)$。下面介绍判定点 q 是否在多边形 P 内的另一个算法,它是周培德提出的。

$Z_{1\text{-}1}$算法

输入 多边形 P 的顶点序列 $p_1(x_1,y_1),\cdots,p_n(x_n,y_n)$(逆时针排列)及边$\overline{p_1p_2},\cdots,$

$\overline{p_{n-1}p_n}, \overline{p_np_1}$；点 $q(x_0, y_0)$。

输出 $q(x_0, y_0)$在 P 内或不在 P 内。

步 1 求$\{y_1, \cdots, y_n\}$的最大、最小值，设为 y_{max}, y_{min}，对应的点为(x_1'', y_{max})，(x_1', y_{min})。求$\{x_1, \cdots, x_n\}$的最大、最小值，设为 x_{max}, x_{min}，对应的点为(x_{min}, y_1')，(x_{max}, y_1'')。

步 2 用点(x_1'', y_{max})和点(x_1', y_{min})分割 P 成左、右两部分 P_1 和 P_2，使得

$P_2 = \{\overrightarrow{(x_1', y_{min})(x_2', y_{min+1})}, \overrightarrow{(x_2', y_{min+1})(x_3', y_{min+2})}, \cdots, \overrightarrow{(x_i', y_{min+i-1})(x_1'', y_{max})}\}$，其顶点集记为 P_2'。

$P_1 = \{\overrightarrow{(x_1'', y_{max})(x_2'', y_{max-1})}, \overrightarrow{(x_2'', y_{max+1})(x_3'', y_{max-2})}, \cdots, \overrightarrow{(x_j'', y_{max+j-1})(x_1', y_{min})}\}$，其顶点集记为 P_1'。

用点(x_{min}, y_1')和点(x_{max}, y_1'')分割 P 成上、下两部分 D_1 和 D_2，使得 $D_2 = \{\overrightarrow{(x_{min}, y_1')(x_{min+1}, y_2')}, \overrightarrow{(x_{min+1}, y_2')(x_{min+2}, y_3')}, \cdots, \overrightarrow{(x_{min+i-1}, y_i')(x_{max}, y_1'')}\}$，其顶点集记为 D_2'。

$D_1 = \{\overrightarrow{(x_{max}, y_1'')(x_{max+1}, y_2'')}, \overrightarrow{(x_{max-1}, y_2'')(x_{max+2}, y_3'')}, \cdots, \overrightarrow{(x_{max+j-1}, y_j'')(x_{min}, y_1')}\}$，其顶点集记为 D_1'。

步 3 **if** $x_0 > x_{max} \vee x_0 < x_{min} \vee y_0 > y_{max} \vee y_0 < y_{min}$

then 输出"$q(x_0, y_0)$在 P 的外部"

else goto 步 4

步 4 **if** $x_0(y_{min+i-1} - y_{min+i}) - y_0(x_i' - x_{i+1}') + (x_i' y_{min+i} - x_{i+1}' y_{min+i-1}) = 0, i = 1, 2, \cdots$

then 输出"$q(x_0, y_0)$在 P_2 上"，即在多边形边界上。

else if $x_0(y_{max+j-1} - y_{max+j}) - y_0(x_j'' - x_{j+1}'') + (x_j'' y_{max+j} - x_{j+1}'' y_{max+j-1}) = 0, j = 1, 2, \cdots$

then 输出"$q(x_0, y_0)$在 P_1 上"，即在多边形边界上。

else goto 步 5

步 5 **if** $(p_i \in P_2' \wedge p_{iy} \neq y_0) \vee (y_{min+i-1} \neq y_0 \wedge y_{min+i} \neq y_0)$

then 在 P_2 中搜索跨越 $y = y_0$ 的线段，即$(y_{min+i-1} < y_0 \wedge y_{min+i} > y_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_i', y_{min+i-1})(x_{i+1}', y_{min+i})}$的左侧$) \vee (y_{min+i-1} > y_0 \wedge y_{min+i} < y_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_i', y_{min+i-1})(x_{i+1}', y_{min+i})}$的右侧$)$并计数该类线段的数目，记为 M_1。

if M_1 为奇数 **then goto** 步 6

else 输出"点 q 在 P 的外部"

else goto 步 6

步 6 **if** $(p_k \in P_1' \wedge p_{ky} \neq y_0) \vee (y_{max+j-1} \neq y_0 \wedge y_{max+j} \neq y_0)$

then 在 P_1 中搜索跨越 $y = y_0$ 的线段，即$(y_{max+j-1} > y_0 \wedge y_{max+j} < y_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_j'', y_{max+j-1})(x_{j+1}'', y_{max+j})}$的左侧$) \vee (y_{max+j-1} < y_0 \wedge y_{max+j} > y_0$

$\wedge q(x_0, y_0)$在$\overrightarrow{(x''_j, y_{max+j-1})(x''_{j+1}, y_{max+j})}$的右侧)并计数该类线段的数目，记为$M_2$。

if M_2为奇数 **then** **goto** 步 7

else 输出“点q在P的外部”

else **goto** 步 7

步 7 **if** $(p'_l \in D'_2 \wedge p'_{lx} \neq x_0) \vee (x_{min+i-1} \neq x_0 \wedge x_{min+i} \neq x_0)$

then 在D_2中搜索跨越$x = x_0$的线段，即$(x_{min+i-1} < x_0 \wedge x_{min+i} > x_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_{min+i-1}, y'_i)(x_{min+i}, y'_{i+1})}$的左侧$) \vee (x_{min+i-1} > x_0 \wedge x_{min+i} < x_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_{min+i-1}, y'_i)(x_{min+i}, y'_{i+1})}$的右侧)并计数该类线段的数目，记为$M_3$。

if M_3为奇数 **then** **goto** 步 8

else 输出“点q在P的外部”

else **goto** 步 8

步 8 **if** $(p'_k \in D'_1 \wedge p'_{kx} \neq x_0) \vee (x_{max+j-1} \neq x_0 \wedge x_{max+j} \neq x_0)$

then 在D_1中搜索跨越$x = x_0$的线段，即$(x_{max+j-1} > x_0 \wedge x_{max+j} < x_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_{max+j-1}, y'_j)(x_{max+j}, y'_{j+1})}$的左侧$) \vee (x_{max+j-1} < x_0 \wedge x_{max+j} > x_0 \wedge q(x_0, y_0)$在$\overrightarrow{(x_{max+j-1}, y'_j)(x_{max+j}, y'_{j+1})}$的右侧)并计数该类线段的数，记为$M_4$。

if M_4为奇数 **then** 输出“点q在P的内部”

else 输出“点q在P的外部”

else **goto** 步 9

步 9 **if** $((p_l \in P'_2 \wedge p_{ly} = y_0) \vee (y_{min+i-1} = y_0 \wedge y_{min+i} = y_0)) \wedge ((p_k \in P'_1 \wedge p_{ky} = y_0) \vee (y_{max+j-1} = y_0 \wedge y_{max+j} = y_0)) \wedge ((p'_l \in D'_2 \wedge p'_{lx} = x_0) \vee (x_{min+i-1} = x_0 \wedge x_{min+i} = x_0)) \wedge ((p'_k \in D'_1 \wedge p'_{kx} = x_0) \vee (x_{max+j-1} = x_0 \wedge x_{max+j} = x_0))$

then 随机选取适当小的$\Delta x > 0, \Delta y > 0$，并且$((p'(x', y') \leftarrow q(x_0 \pm \Delta x, y_0)) \vee (p'(x', y') \leftarrow q(x_0, y_0 \pm \Delta y)) \vee (p'(x', y') \leftarrow q(x_0 \pm \Delta x, y_0 \pm \Delta y)))$
$\wedge$ $((\overrightarrow{qp'}$与$\overrightarrow{(x'_i, y_{min+i-1})(x'_{i+1}, y_{min+i})}$不相交$) \vee (\overrightarrow{qp'}$与$\overrightarrow{(x''_j, y_{max+j-1})(x''_{j+1}, y_{max+j})}$不相交))

步 10 $q \leftarrow p'$，**goto** 步 5

定理 1-1 算法$Z_{1\text{-}1}$正确地判定点q是否在多边形内部。算法$Z_{1\text{-}1}$至多需要$O(n)$次四则运算和$O(n)$次比较。

证明 如果点$q(x_0, y_0)$在多边形P的内部，那么射线$y = y_0$与P的边第一次相交之后便穿过多边形的边界而进入多边形的外部。如果该射线延伸后再与多边形的边相交(第二次相交)，射线便又进入多边形内部。依次类推。这就是说，如果点q在多边形P的内部时，过点q的四方向的射线必与多边形的边相交奇数次。同理，如果点q在多边形的外部时，至少有一条射线与多边形的边必相交偶数次(包括 0 次)。这是该算法的依据。

算法的第 9、10 步处理几种特殊情况，即射线 $y=y_0$ 穿过多边形顶点或与一条边重合。此时便改为考查点 $p'(x',y')$ 是否在多边形 P 的内部。由于算法 $Z_{1\text{-}1}$ 的第 9 步保证了 $\overrightarrow{qp'}$ 与多边形的任一条边不相交，所以点 p' 与 q 同在多边形的内部或外部。这样便可以把"判定点 q 是否在多边形内部"的问题转化为"判定点 p' 是否在多边形内部"的问题。因此该算法能够正确地确定点 q 是否在多边形内部。

在一般情况下，算法 $Z_{1\text{-}1}$ 于步 5、步 6、步 7 和步 8 终止的可能性较大，而经步 9、步 10 后的循环的可能性很小。

算法的第 1 步求最大、最小值，耗费 $2\lceil 3n/2-2\rceil$ 次比较。第 2 步需要线性次比较，这是因为输入时多边形的边及顶点已按逆时针方向排列。第 2 步的工作是用 4 个点断开这种排列，即在已排序点列中寻找点 $(x_1',y_{\min})$，$(x_{\max},y_1'')$，$(x_1'',y_{\max})$，$(x_{\min},y_1')$。第 3 步耗费常数时间。第 4 步判定点 q 是否在多边形 P 的边界上，至多需要 $4n$ 次乘法和 $5n$ 次加减法。

第 5，6，7，8 步分别计算 M_1，M_2，M_3 和 M_4，至多需要线性次比较和线性次判定点在有向线段的哪一侧，而判定一个点在有向线段的哪一侧只需要 12 次乘法，所以耗费线性次比较和线性次乘法便可计算出 M_1，M_2，M_3 与 M_4 的值。

算法 $Z_{1\text{-}1}$ 的第 9 步处理几种特殊情况，需要线性次判定两线段是否相交，而每次判定两线段是否相交仅需 10 次乘法，所以第 9 步至多需要 $O(n)$ 次乘法。只要合理地选取 Δx 或 Δy，转入第 5 步的循环次数便可得到有效控制。比如，由点 $q(x_0,y_0)$ 引出的水平射线穿过点 $p_i(x_i,y_i)$，这表明 $y_0=y_i$。这时可以选取 p' 的横坐标为 x_0，纵坐标为 $y_0+\Delta y$，$\Delta y=|(y_{i+1}-y_i)/2|$，$y_{i+1}$ 是点 p_{i+1} 的纵坐标，并且 p' 与 q 同属于多边形的内部或外部，即 $\overline{qp'}$ 与多边形的边不相交。在这种情况下，转入第 5 步的循环一般只要进行一次即可判定点 q 是否在多边形内部。

总之，算法 $Z_{1\text{-}1}$ 在最坏情况下至多需要线性次比较和线性次乘法及线性次加减法。

证毕。

假设 P 是凸 n 边形，取 P 的任意三个顶点作为三角形的顶点，作该三角形的形心，设为 z。显然，z 在 P 的内部。然后从 z 出发，作过 P 各顶点的射线，有 n 条射线，这些射线将平面划分成为 n 个楔形。P 的每条边又将相应楔形分为两部分：P 的内部与 P 的外部。以 z 为坐标原点，由于射线以角次序出现，所以可以用对分查找判定点 q 所在的楔形。然后再判定 q 位于相应 P 边的哪一侧，便可确定 q 是否在 P 内。这种方法的预处理包括寻找 P 的内点 z，以及组织数据使之适合于对分查找。由于事先给定了序列 $p_1,p_2,\cdots,p_n$，所以用 $O(n)$ 时间能建立适于对分查找的二叉树。具体查找 q 是否在 P 内只需要 $O(\log n)$ 时间。

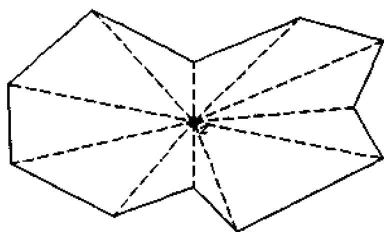


图 1-3　一个星形多边形

为了能进行对分查找，P 的顶点一定要关于点 z 按顺序出现。当 P 为星形多边形时，如图 1-3 所示，关键是要找到点 z，使得 $\overline{zp_i}(i=\overline{1,n})$ 整个在 P 内，这样的 z 点的集合称为 P 的核。可以证明，在线性时间

内可以确定多边形 P 的核(若非空)。这样,选择 z 点并产生楔形的时间限界不会超过线性时间。因此,用 $O(\log n)$时间和 $O(n)$空间能回答 n 个顶点的星形多边形的包含问题,其预处理时间为 $O(n)$。

当 P 是任意简单多边形时,可以先将 P 分解为若干个凸多边形,然后再用上述方法回答多边形的包含问题。

1.1.2 确定点 q 在平面剖分中的位置

平面图总可以嵌入平面,它的边映射为直线段,这种嵌入的图称为平面直线图,记为 G。平面直线图 G 确定平面的一种剖分,若 G 的顶点度数不小于 2,则该图的有界区域是简单多边形。假设 G 是连通的。

对平面剖分中的每个子区域逐一检查,可以确定点 q 所在的子区域。但这种方法耗费的时间多,不可取。为了加快定位点 q 的速度,一种有效的方法是将平面直线图 G 的所有有界区域组织成二叉树结构,然后进行对分查找。下面介绍几种方法。

1. 水平长条方法

给定平面直线图 G,通过 G 的每个顶点画一条水平线,如果 G 有 n 个顶点并且没有两个顶点的 y 坐标相同,则平面被划分成 $n+1$ 个水平长条,如图 1-4 所示。依据顶点的 y 坐标分类这些水平长条,然后执行对分查找,耗费 $O(\log n)$查找时间可以确定点 q 所在的水平长条。

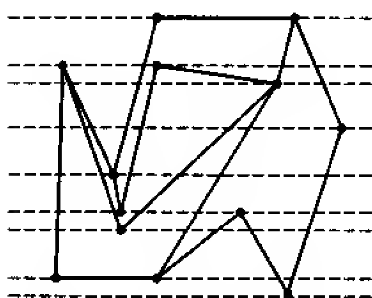


图 1-4 平面直线图顶点确定的水平长条

每个水平长条均被 G 的某些边划分成梯形或三角形。这些边可以从左至右排序,也就是说,位于同一水平长条中的梯形或三角形可以从左至右排序,然后利用对分查找,耗费 $O(\log n)$查找时间可以确定点 q 所在的梯形。

下面讨论这种方法所需要的预处理时间和空间。显然,每个水平长条可能有 $O(n)$条线段,分类这些线段耗费 $O(n\log n)$时间,分类 n 个水平长条中的线段所需要的时间为 $O(n^2\log n)$,存储空间为 $O(n^2)$。

降低预处理时间 $O(n^2\log n)$的一种方法是采用由底向顶的水平扫描方法,G 的每个顶点是事件点。在事件点处,扫描线状态要修改并读出,该读出恰好是下一个水平条中线段的序列。一个水平条中可能有 $O(n)$条线段,将这些线段组成二叉树,耗费 $O(n)$时间。对所有水平条都如此处理,所以预处理时间是 $O(n^2)$。存储空间仍为 $O(n^2)$。

上述方法的优点是查询时间少,仅为 $O(\log n)$,而缺点是,所需要的预处理时间达到 $O(n^2)$并且存储为 $O(n^2)$。下面介绍存储为 $O(n)$,查询时间为 $O(\log^2 n)$的方法。

2. 链方法

引进几个概念,包括链、单调链、链的单调完全集、正则图等,并同时介绍一些算法。

链 $C=(u_1,\cdots,u_n)$是一个具有顶点集$\{u_1,\cdots,u_n\}$和边集$\{\overline{u_iu_{i+1}}\mid i=\overline{1,n-1}\}$的平面直线图。可以把链理解为一条折线。给定点 q，判定点 q 在 C 的哪一侧，与测试多边形是否包含点 q 基本上是同一个问题。如果对链 C 加以适当的限制，则可以更容易地判定点 q 在 C 的哪一侧，因此下面引入单调链的概念。

若正交于 l 的一条直线恰好交 C 于一点，则称链 C 关于直线 l 是单调的，如图 1-5 所示。链 C 的顶点在 l 上的正交投影 $l(u_1),\cdots,l(u_n)$ 是有序的，因此可以组织成二叉树结构，故能用对分查找把询问点 q 在 l 上的投影 $l(q)$定位在唯一的区间$(l(u_i),l(u_{i+1}))$中，然后再判定 q 位于线段 $\overline{u_iu_{i+1}}$的哪一侧。这个过程耗费 $O(\log n)$时间便可确定点 q 在链 C 的哪一侧。

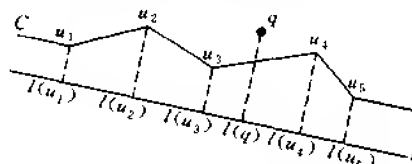


图 1-5　链 C 关于直线 l 是单调的

利用单调链能快速定位询问点 q。这就提供了一种解决问题的方式，即先将平面直线图 G 转化成多条单调链集，然后再定位点 q。下面引进 G 的链的单调完全集的概念。

设链集 $B=\{C_1,\cdots,C_m\}$中的每条链 $C_j(j=\overline{1,m})$关于直线 l 单调，并且满足条件(1) $\bigcup_{j=1}^{m}C_j\supset G$；(2) $C_i,C_j\in B,u_{i1},\cdots,u_{ip}$是 C_i 的顶点，如果 $u_{i1},\cdots,u_{ir},u_{ir+k},\cdots,u_{ip}$不是 C_j 的顶点，则这些顶点位于 C_j 的同一侧。称集合 B 为 G 的链的单调完全集。注意，G 的一条边能属于 B 中的多条链，另外，条件(2)意味着 B 中的链是有序的，因此对 B 可以应用对分查找，其中基本操作不是比较，而是判断点在链的哪一侧。因此，如果 B 中有 m 条链，且最长的链有 p 个顶点，则查找所需要的时间至多为 $O(\log p\cdot\log m)$。如果 G 有 n 个顶点，则查找时间的最坏情况的上界为 $O(\log^2 n)$。

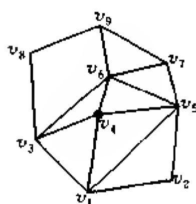


图 1-6　正则平面直线图的一个例子

设平面直线图 G 的顶点为 $v_1,\cdots,v_n$，其中顶点下标与其坐标满足：$i<j$ 当且仅当 $v_{iy}<v_{jy}$，或者若 $v_{iy}=v_{jy}$，则 $v_{ix}>v_{jx}$。若有整数 $i<j<k$，使得$\overline{v_iv_j}$和$\overline{v_jv_k}$是 G 的边，则顶点 v_j 称为正则的。如果 G 的每个顶点都是正则的(v_1,v_n 除外)，则称 G 为正则的。图 1-6 是正则图的一个例子。可以证明正则图容许有关于 y 轴单调的一个链的完全集(y 轴即 l 直线)。

当 $i<j$ 时，规定 v_i 与 v_j 之间的边为$\overline{v_iv_j}$。因此可以用 $\mathrm{IN}(v_j)$和 $\mathrm{OUT}(v_j)$表示 v_j 的进入和外出的边集，并且 $\mathrm{IN}(v_j)$中的边按逆时针排列，而 $\mathrm{OUT}(v_j)$中的边按顺时针排列。由于 G 是正则的，所以这两个集合是非空的(v_1，v_n 除外)。然后利用由 v_2 至 v_{n-1}的第一遍扫描给各边赋权值及 v_{n-1}至 v_2 的第二遍扫描再次给各边赋权值，并且使各点的出入度数相等。最后依据各边权值生成 G 关于 y 轴单调的链的完全集。

上述方法是 Even 提出的，该方法是在假设 G 为正则的条件下完成构造链的完全集的工作，剩下的任务是如何把任意一个平面直线图变换成正则平面直线图。利用水平扫描线来回扫描，并补充某些线段使非正则顶点正则化的方法可以解决这个问题。这种方法的时间复杂性为 $O(n\log n)$，空间为 $O(n)$。

由于 n 个顶点的平面直线图可以具有 $O(\sqrt{n})$条链，每条链有 $O(\sqrt{n})$条边，故查找

时间可以达到上界 $O(\log^2 n)$。如果将链自左向右进行编号，然后组织成二叉树，则查找结构的空间耗费为 $O(n)$。

总之，链方法在预处理时间 $O(n\log n)$、空间复杂性 $O(n)$、查找时间 $O(\log^2 n)$ 内能完成 n 个顶点平面剖分中的点定位。

对于 n 个顶点的单调多边形来说，用 $O(\log n)$ 时间能判定单调多边形中的包含问题。

下面介绍另一个由周培德提出的算法，该算法不仅将平面直线图 G 变成正则平面直线图，而且构造了链的完全集。这就是说，上述两项工作（平面直线图变成正则平面直线图；构造链的完全集）由一个算法完成。

$Z_{1\text{-}2}$算法

输入　平面直线图 G 的顶点集 $V=\{v_1,v_2,\cdots,v_n\}$ 和边集 $E=\{e_1,e_2,\cdots,e_m\}$，与顶点 v_i 关联的边按逆时针方向排列，记为 $e^{i_1},e^{i_2},\cdots,e^{i_{j+1}},\cdots,e^{i_{v_i}}$。

输出　图 G 的完全单调链集 $\mathscr{C}=\{C_1,C_2,\cdots,C_r\}$。

步 1　求顶点集 V 的凸壳，设 $v_1',v_2',\cdots,v_k'$ 是凸壳顶点（凸壳的边不一定是 E 中的边）。

步 2　求凸壳直径，设 v_1',v_l' 是直径的两个端点。$\overline{v_1'v_l'}$分凸壳为两则 C^1 与 C^2。

步 3　各点 $v_i(i=\overline{1,n})$垂直投影到$\overline{v_1'v_l'}$上，并按投影点的 x（或 y）坐标排序。v_i（直径，x）表示 v_i 在直径上的投影点的 x 坐标；v_{i+a}表示与 v_i 连续相邻关联的第 a 个点；v_{i-1},v_{i+1}是与 v_i 直接相邻关联的 2 个点，如图 1-7 所示。

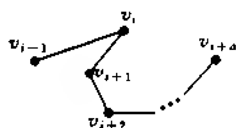


图 1-7　与 v_i 相邻关联的点

步 4　按照"e^{i_j}与 $e^{i_{j+1}}$分别为 v_i 的入边及出边，并且 e^{i_j}的另一端点、v_i、$e^{i_{j+1}}$的另一端点在$\overline{v_1'v_l'}$上的投影点的 x 坐标呈递增序排序"的规则，连续选边及顶点（即累接方法）构成链。删去已选边及度数为 0 的顶点，更新 v_i 关联边的编号。由 v_1' 或凸壳顶点或度数不为 0 的顶点开始，并从凸壳一侧 C^1（或两侧 C^1，C^2）逐层向内按上述规则找出链 $C_1,C_2,\cdots,C_r$。直至剩余边集与剩余顶点集均为空。

步 5　**if**　$C_i(2\leqslant i\leqslant r)$不以 v_1',v_l' 为端点　**then**　延伸 C_i，使以 v_1',v_l' 为端点而且 C_i 与 C_{i+1},C_{i-1}互不交叉，但可能有共同的边。

步 6　检查不能延伸（累接）到 v_1',v_l' 的链的端点 v_i。

if　$v_{i-1},v_{i+1},\cdots,v_{i+k-1}$（直径，$x$）＜（或＞）$v_i$（直径，$x$）$\wedge$ v_{i+k}（直径，x）＞（或＜）v_i（直径，x）

then　连接 v_i 与 v_{i+k}，$\overline{v_iv_{i+k}}$加入 E，按步 4 规则构造链。

定理 1-2　给定任意具有 n 个顶点、m 条边的平面直线图 G，$Z_{1\text{-}2}$算法正确地求出 G 的完全单调链集 $\mathscr{C}$，且算法的复杂性为 $O(n\log n)$次比较和 $O(n)$次乘法。

证明　算法 $Z_{1\text{-}2}$的步 1 求出凸壳的顶点 $v_1',v_2',\cdots,v_k'$。然后步 2 求凸壳直径$\overline{v_1'v_l'}$，$\overline{v_1'v_l'}$将凸壳分成两部分 C^1 与 C^2。C^1 为寻找链集 $\mathscr{C}$ 提供了起始位置。当组成 C^1 的边都属于 E 时，$C_1=C^1$；否则由 v_1' 开始，沿与 v_1' 关联的并最靠近 $\overline{v_1'v_2'}$的边，按步 4 的规则选边与顶点，

进入 C_1。确定 C_1 之后，删去进入 C_1 的边及度数为 0 的顶点，并修改 C_1 中顶点关联边的编号。重复步 4，直至剩余边集、剩余顶点集为空。由于每条边一经选入某链 C_i，它便立即被删去，因此 C_i 与 $C_j(i\neq j)$ 不可能有共同的边。又由于选择 v_i 的入边与出边时，要求 e'_i 的另一端点，v_i 和 e'_{i+1} 的另一端点在 $\overline{v'_1v'_i}$ 上的投影点的 x 坐标保持递增序，所以找出的链 C_i 对于某直线(即凸壳直径 $\overline{v'_1v'_i}$)始终保持单调性。步 5 的工作使所有的链都以 v'_1 和 v'_i 为端点，某些链有共同的边。由于 C_1 对直径是单调的，而延伸(累接) $C_i(2\leqslant i\leqslant r)$ 不会破坏单调性，所以延伸(累接)后的 $C_i(2\leqslant i\leqslant r)$ 对直径是单调的。若 G 中有一些点在 $\overline{v'_1v'_i}$ 上的投影点相同，则可将 $\overline{v'_1v'_i}$ 旋转一个小的角度，使其投影点不同或按步 4 的规定也可以得到不同的单调(对于 $\overline{v'_1v'_i}$ 来说)链集(此时某些边垂直于 $\overline{v'_1v'_i}$)。步 6 处理某些特殊的顶点，在 G 中添加某些特殊的边，使构造的链都始于 v'_1 而终于 v'_i，并保持单调性。总之，算法正确地求出 G 的完全单调链集。

算法的步 1 求凸壳耗费 $O(n\log n)$ 次比较。步 2 求凸壳直径需要 $O(n)$ 次乘法和 $O(n)$ 次比较。步 3 求垂直投影点需要 $O(n)$ 次乘法，再用 $O(n\log n)$ 次比较可以完成投影点的 x 坐标排序。步 4 和步 5 耗费 $O(|E|)$ 次比较可处理特殊点。因此算法的时间复杂性为 $O(n\log n)$ 或 $O(|E|)$ 次比较和 $O(n)$ 次乘法。　　证毕。

上述 $Z_{1\text{-}2}$ 算法采用了与 Even 算法不同的方法解决了平面直线图正则化问题，并求得 G 的完全单调链集，其复杂性与 Even 算法(只完成构造链的完全集工作)有相同的阶。本文算法不仅将 G 正则化，而且求得 G 的完全单调链集，另外还确定了 G 对哪条直线单调。一个算法完成了三件工作，因而优于 Even 算法。

图 1-8 与图 1-9 是 $Z_{1\text{-}2}$ 算法应用举例，对图 1-8(a)、图 1-9(a)执行 $Z_{1\text{-}2}$ 算法，得到图 1-8(c)、图 1-9(c)所示完全单调链集。

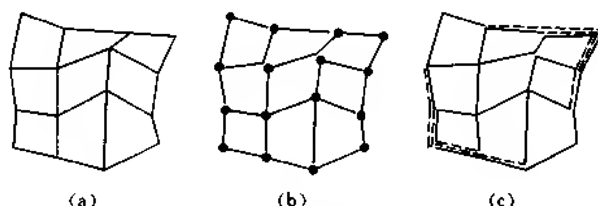


图 1-8　$Z_{1\text{-}2}$ 算法举例

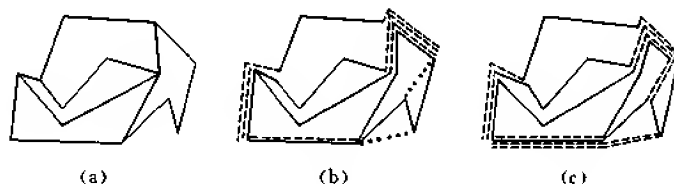


图 1-9　$Z_{1\text{-}2}$ 算法举例

3. 三角剖分加细方法

三角剖分加细方法是Kirkpatrick 提出的。假设 n 个顶点的平面直线图 G 是一个三角剖分，它最多有 $3n-6$ 条边。由 G 开始，构造一个三角剖分序列，然后将该序列组织成便于查找的结构形式 T，最后确定询问点 q 所在的三角形域。

构造 G 的三角剖分序列 $S_1, S_2, \cdots, S_{h(n)}$ 的算法如下：

步 1 $i \leftarrow 1, S_i \leftarrow G$

步 2 删去 S_i 的非邻接的非边界顶点(设为 $q_1^i, q_2^i, \cdots, q_m^i$)的集合以及它们关联的边，$G$ 内产生若干个新的多边形 $p_j(j=\overline{1,m})$

步 3 三角剖分 $p_j(j=\overline{1,m})$

步 4 重复执行步 2 和步 3，直至 $S_{h(n)}$ 内没有顶点。

上述算法每循环一次，产生一批新的三角形。构造 S_i 时，产生的三角形记为 R_j(或简记 j)，有的三角形可出现在多个三角剖分中。为了便于查找，要建立一个查找数据结构 T，它是一个有向的非循环图，其结点代表三角形，如图 1-10 所示。图 1-10(a)中，表示三角剖分序列，画圈的顶点表示将要删去的顶点。在图 1-10(b)中，T 以分层方式表示，每层对应于一个三角剖分，每个结点代表一个三角形。

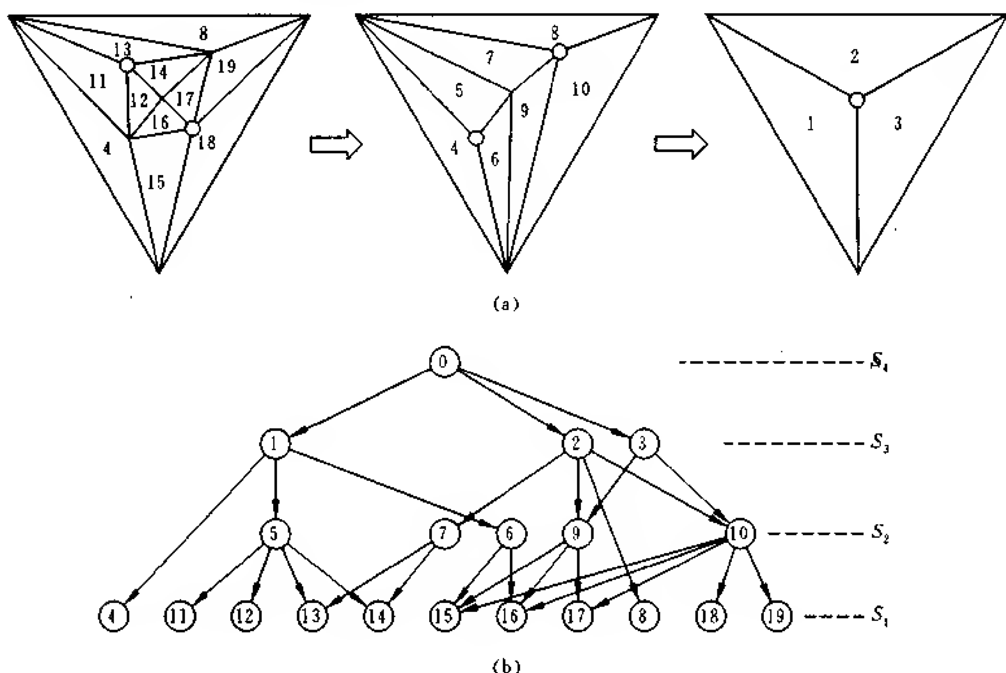


图 1-10

(a) 一个三角剖分序列 (b) 与(a)对应的有向非循环图

建立数据结构 T 之后，沿 T 的路径(从根至叶)进行搜索，原始操作是"包含在一个三角形中"(用 $O(1)$时间可以完成)。开始时，定位 q 是否在 $S_{h(n)}$ 中，然后沿一条路径向下，最

后在属于 S_1 的一个三角形处终止。假设搜索进行到 S_i，这表明点 q 位于 S_i 的某个三角形 R_j 内，由于 S_{i-1} 是 S_i 的加细，所以在 R_j 的进一步细分(更小的三角形)中测试是否包含点 q。表 $L(u)$ 存放的结点 u 为根的所有子孙，并用 Trian(u)表示对应于结点 u 的三角形，则查找算法描述如下：

```
Procedure point location
begin
  if q∉Trian(根) then 点 q 属于无界域
  else begin
          u←根;
          while L(u)≠∅ do
                for 每个 v∈L(u) do
                      if q∈Trian(v) then u←v
          print u
      end
end
```

可以证明结构 T 中，$h(n)=O(\log n)$，存储结点耗费 $O(n)$ 空间。另外，删去顶点时，依据"度数小于 $k(=12)$ 的非邻接顶点"选择顶点集合作为删去顶点。构造 n 个顶点的三角剖分图 G 需要 $O(n\log n)$ 时间，这也是三角剖分加细方法的预处理时间。由于 $h(n)=O(\log n)$，而处理 T 中每个结点的耗费为 $O(1)$ 时间，故查找时间为 $O(\log n)$。因此，三角剖分加细方法耗费预处理时间 $O(n\log n)$、查找时间 $O(\log n)$ 和存储 $O(n)$ 可以完成 n 个顶点平面剖分中的点定位。

4. 梯形方法

梯形方法是水平长条法的进一步发展，水平长条法要避免长边(即与多个水平条相交的边)，但长边对于梯形法却有利。虽然梯形法的查找时间较水平长条法多一些，但梯形法易于实现，并且存储耗费也低于水平长条法。下面先介绍两个概念，然后描述梯形方法并列出复杂性。

梯形是一个四边形，其中有两条水平边，而且可以是有界的或单侧无界的或双侧无界的；另外两条边(如果存在的话)是平面直线图 G 的边(的一部分)，并且没有 G 的边穿过该四边形的内部。

梯形方法不要求三角剖分 G，甚至不要求边是直线的。假设平面直线图 G 的顶点集 V 按其 y 坐标递增序分类，而边集 E 按偏序关系 $\prec$ 来排序。

设 e_1 和 e_2 是 E 中的两条边，$e_1 \prec e_2$(读作"e_1 在 e_2 的左边")表示有一条水平线与两条边 e_1、e_2 相交，其交点分别为 a_1、a_2，并且 a_1 在 a_2 的左边。

构造查找数据结构的过程是一次处理一个梯形 R，即把 R 划分为更小的梯形。开始时用过 G 的一个顶点 p_i 的水平线把 R 分割为下片 R_1 和上片 R_2，p_i 的纵坐标 y_i 是 R 内 G 顶点集合中的中值，即 $y=y_{med}$。然后从左到右扫描与 R 相交的 G 的边串，且分为两串，分别属于 R_1 和 R_2。扫描碰到 G 的边，此边就是新梯形的右边界，如图 1-11 所示。

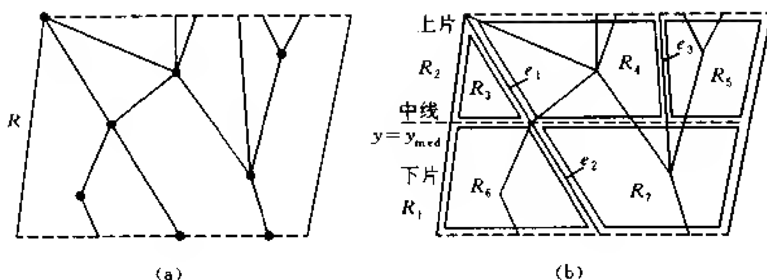


图 1-11

(a) 梯形 R (b) R 被划分为小梯形 R_3、R_4、R_5、R_6 和 R_7

与图 1-11(b)对应的梯形查找数据结构，如图 1-12 所示。$T(R)$表示对梯形 R 进行对分查找的树结构。其结点分为两类：若关联的测试对应于一条水平线，则为三角形结点；若关联的测试对应于包含 G 的一条边，则为圆形结点。显然，$T(R)$的根结点是一个三角形结点，此结点对应于 G 的一个顶点，该顶点的纵坐标是梯形 R 中 G 顶点纵坐标的中值。在查找树中有 $n-2$ 个三角形结点。

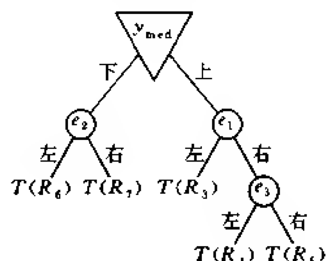


图 1-12 与图 1-11(b)对应的梯形查找数据结构

构造梯形查找数据结构的步骤如下：

步 1 计算 R 内平面直线图 G 顶点的中值纵坐标。

步 2 划分上片和下片为梯形，并对上、下片产生串 $U_i(i=1,2)$，U_i 的项是边和树(比如图1-12中，$U_1=T(R_6)e_2T(R_7)$，$U_2=T(R_3)e_1T(R_4)e_3T(R_5)$)。

步 3 均衡两个串 U_1 和 U_2。

用经典的计算中值算法在 $O(|V|)$时间内可以完成步 1。但另一种较简单的方法是，将梯形内 G 的顶点按 y 坐标递增序排列并存于一个数组中，然后一次访问能找到此数组的中值，并对边序列扫描两次。第一次扫描时，用赋予梯形的名称来标记每个顶点，并构造每个生成梯形的顶点数组。第二次扫描中，产生串 U_1 和 U_2。最后把树和边的交错串 $U=T_1e_1T_2e_2\cdots e_{h-1}T_h$ 插入均衡树。梯形中 G 的顶点数作为对应树 T_j 的权值 $W(T_j)$。U 的权 $W(U)=\sum_{j=1}^{r}W(T_j)$。

利用对 $W(U)$的归纳法，可以证明该均衡树的深度 $\delta(U)$ 至多为 $3\log W(U)+\lceil\log n\rceil+3$。对于平面直线图 G 来说，$W(U)\leqslant n$，因此 G 的查找树的深度至多为 $4\lceil\log n\rceil+3$，这就是说，梯形法的查询时间至多为 $4\log n$ 次测试。另外，计算中值耗费线性时间，而均衡工作耗费 $O(n\log n)$时间，所以构造查找数据结构需要 $O(n\log n)$时间。对于每条边段，查找数据结构有一个圆形结点，且对于 G 的每个顶点有一个三角形结点，因此梯形法的空间耗费为 $O(n\log n)$。总之，梯形方法耗费 $O(n\log n)$存储和预处理时间，至多用 $4\log n$ 次测试能在 n 个顶点的平面剖分中定位一个点。

1.2 范围查找问题

范围查找问题是点定位问题的对偶问题。指定 d 维空间中一个域 D(称为询问域),范围询问是报告包含在 D 中点集 S 的子集 S',或者计数位于 D 内点集 S' 中的点数。前者使用集合并运算而后者使用加法运算。

d 维空间中的每一个维表示与该空间中点关联的一种信息,$d \geq 2$ 时,对域 D 的查找称为 d 重查找。比如查找某单位年龄在 25 至 35 岁之间,工资在 1000 元至 2000 元之间的人员名单,便是 2 重查找。

询问域 D 是根据询问的性质来确定的,大多数询问域是矩形域、超矩形域(即不同坐标轴上区间的笛卡尔乘积,又称为正交询问)或者圆域。D 为圆域时,使用了邻近概念。为了回答范围询问,必须要建立 d 维空间中点集 S 的查找数据结构,这种数据结构分为静态的(一旦建立起便不再修改)和动态的(对项可以删除或插入)两种。建立这种数据结构将要耗费预处理时间和存储空间,此外,询问时间也是一种要考虑的时间开销。当静态的查找数据结构建立之后,主要考虑询问时间和存储。因此用耗费对(存储、询问时间)来描述范围查找方法的特征。对于实际问题的不同要求,可以选择相异的查找方法,使其在存储和询问时间之间进行折衷。

询问时间依赖于点集 S 的大小 n 和包含在 D 内的子集 S'(又称存取集合)的大小,还依赖于询问类型(即集合并运算或加法运算)。报告型询问要报告 S' 的 k 个成员的名称,而计数型询问则报告一个整数(S' 的基数 k)。处理一个询问时有两种操作类型:

(1) 查找,产生存取集合项的操作(通常为比较序列);

(2) 检索,收集询问响应的操作(对于报告型询问,是检索存取子集)。

在计数型询问中,计算工作是指查找,此时,最坏情况询问时间是点集 S 的大小 n 和维数 d 的函数 $f(n,d)$。报告型询问中,计算工作包括查找和检索两种操作,而且后者部分以 S' 的大小 k 为下界。若要求询问存取 S' 的成员,则两种类型询问的查找具有相同的耗费。如果允许用比目标 S' 稍大的母集代替存取集,则有可能减少存储耗费并增加检索工作部分的查找操作耗费,这种查找称为过滤查找。因此,报告型询问的询问时间的一个上界可以表示为 $O(f(n,d)+k \cdot g(n,d))$;若存取的为检索集,则其中两项分别解释为查找时间和检索时间。

由于输出长度和 k 成比例,所以 $\Omega(k)$ 是检索时间的平凡下界。另外,利用二叉判定树模型可以证明查找时间的一个下界是 $\Omega(d\log n)$,而动态询问系统查找时间的一个下界是 $\Omega((\log n)^d)$。最后,$\Omega(nd)$ 是查找数据结构所需存储空间的一个平凡下界。

下面介绍几个范围查找算法。范围查找的最简单实例是一维范围查找问题:假设 n 个点分布在 x 轴上,询问范围是区间 $[a,b]$(称为 x 范围)。利用对分查找方法(即平分被查找的有序集)可以有效地进行查找。这种方法采用均衡二叉树的数据结构,其叶结点另外被连接作为反映横坐标次序的一个表。查找和检索时分别访问树和表。该方法的询问时间 $\theta(\log n+k)$ 和存储 $\theta(n)$ 都是最优的。

1.2.1 多维二叉树(k-D 树)的方法

给定平面上 n 个点的点集 S,交替使用平行于 x 轴或 y 轴的直线将平面平分成两个矩形(有界的或者无界的),这些直线过 S 中的点,并且使直线两侧 S 内点的数目近似相等。这里的矩形是 x 区间$[x_1,x_2]$和 y 区间$[y_1,y_2]$的笛卡尔乘积$[x_1,x_2]\times[y_1,y_2]$,其中 $x_i,y_i(i=1,2)$可以为$-\infty,+\infty$。显然,这是利用分治思想设计的一种方法。

上述划分平面的过程可以与一棵二维二叉树 T 联系起来。划分过程中所产生的矩形分别赋给 T 中相应的结点,即结点 v 与矩形 $R(v)$关联,且点集 $S(v)\subseteq S$ 在矩形 $R(v)$中。也就是说,T 中结点 v 关联于 $S(v)$的一个选取点 $p(v)$以及一条通过 $p(v)$且平行于一个坐标轴的切割线。

开始时,定义 T 的根并设置 R(根)为整个平面,S(根)$=S$;然后确定点 $p\in S$ 使 $x(p)$ 是 S(根)的点的 x 轴坐标的中值。过点 p 的直线(平行于 y 轴)把 S 分为两个大小近似相等的集合(平面被平分为两个半平面,用 R_1 和 R_2 表示),R_1 和 R_2 被赋给根的两个子结点。继续分割下去,当该过程达到一个不包含任何点的矩形时,终止该分割过程,对应的结点是 T 的叶结点。上述分割过程及对应的二维二叉树 T(即 2-D 树)如图 1-13 所示。

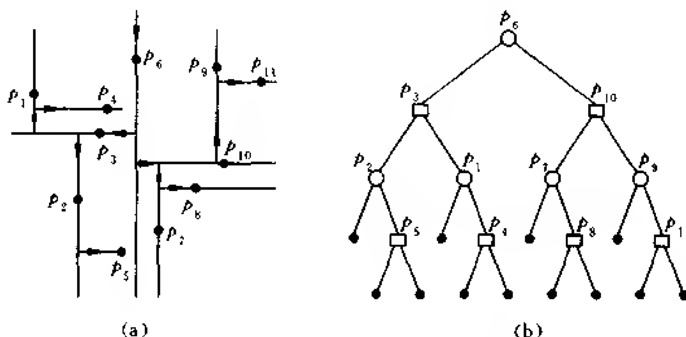


图 1-13 二维二叉树查找方法的说明

在图 1-13 中,$n=11$,二叉树中有三种不同类型的结点:圆形结点表示垂直切割线的非叶结点;方形结点表示水平切割线的非叶结点;而实心结点表示叶结点。

范围查找中 2-D 树的用法如下:当关联于 T 的结点 v 的矩形域 $R(v)$和矩形范围 D 的交非空时,过 $p(v)$的直线 $l(v)$把 $R(v)$分割为两个矩形 R_1 和 R_2。若 $D\cap R(v)$整个包含在 $R_i(i=1,2)$中,则继续查找一个(区域,范围)对(R_i,D)。另一方面,若由 $l(v)$分割 $D\cap R(v)$,这表明 $l(v)$和 D 的交非空,所以 D 可包含 $p(v)$。因此,首先要测试 $p(v)$是否在 D 的内部。如果在 D 的内部,则把它放入检索集合,然后继续查找两个(区域,范围)对(R_1,D)和(R_2,D)。若查找工作达到叶结点,则终止范围查找。T 的结点 v 带有三个参数$(p(v),t(v),M(v))$,其中点 $p(v)$已被定义,另外两个参数共同确定直线 $l(v)$,即 $t(v)$表明 $l(v)$是水平的,或者垂直的。而且在第一种情况下,$l(v)$是直线 $y=M(v)$,而在第二种情况下,$l(v)$是直线 $x=M(v)$。算法把检索的点放在表 U 中(开始时 U 为空)。$D=[x_1,x_2]\times[y_1,y_2]$表示询问范围。查找算法描述如下:

```
procedure  SEARCH(v,D)
begin
  if  t(v)=垂直的  then  [l,r]←[x1,x2]
  else  [l,r]←[y1,y2];
    if  l≤M(v)≤r  then
      if  p(v)∈D  then  U⇐p(v);
    if  v≠叶结点  then
      if  l<M(v)  then SEARCH(Lson[v],D);
      if  M(v)<r  then SEARCH(Rson [v],D)
end
```

对图 1-13 所示例子的查找过程如图 1-14 所示，图 1-14(a)中虚线表示询问范围 D，图 1-14(b)中虚线表示查找算法执行的 T 的结点的访问。在查找访问分叉的结点处(比如 $p_6,p_3,p_2,p_4,p_{10},p_7,p_8$)要进行点是否包含在 D 内的测试。星号(＊)标记的结点是测试成功的结点，且取出该点。图中的检索集合为$\{p_3,p_4,p_8\}$。

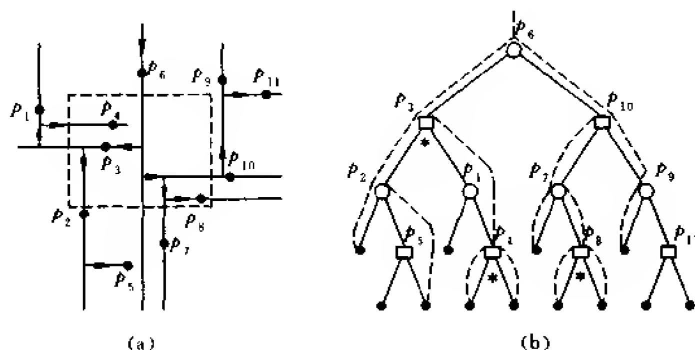


图 1-14　图 1-13 所示例子的查找过程

设执行查找算法所需要的时间为 $T(n)$，初次平分点集 S(求中值)耗费 $O(n)$时间，S 被分成 S_1 和 S_2，它们分别有大约 $\left\lfloor \frac{n}{2} \right\rfloor$ 个点，然后递归求中值并分割子集，所以 $T(n)$满足下述递归关系式

$$T(n) \leqslant 2T\left(\frac{n}{2}\right) + O(n)$$

求解该递归关系式得到 $T(n)=O(n\log n)$，这是预处理时间。在 T 中每个结点处查找算法耗费常数时间，因此询问时间与查找算法访问 T 的结点数成比例。若 $p(v)$被检索(生成的结点)，则在结点 v 处利用了此时间；否则，结点是非生成的。T 的每个结点 v 对应于一个矩形 $R(v)$。询问范围 D 和 $R(v)$的交有几种不同情况，只要区分这些情况中哪些是生成的，哪些可能是非生成的，便可以证明，高度 m 的 T 的子树中非生成结点数为 $O(\sqrt{n})$，这表明二维中询问时间为 $O\left(n^{\frac{1}{2}}\right)$。另外，存储耗费为 $\theta(n)$。

上述方法可以推广到 d 维($d>2$)，这时要考虑正交于坐标轴的切割超平面，并选取切割超平面的定向准则。用多维二叉树(k-D 树)作为 d 维空间划分的模型。可以证明，对

于 $d \geqslant 2$，用 k-D 树方法，耗费 $\theta(dn\log n)$ 预处理时间，$\theta(dn)$ 存储，询问时间 $O(dn^{1-1/d}+k)$ 可以完成 d 维 n 个点集合的范围查找。这表明 k-D 树方法是一个 $(dn, dn^{1-1/d})$ 算法，但它的最坏情况的效率低，即可能生成满二叉树，因此要研究其他方法。

1.2.2 直接存取方法

该方法（又称矩形方法）的基本思想是预先计算所有可能范围查找询问的解并保留结果，然后进行一次存储存取便可以完成查找，所耗查找时间为 $O(1)$。这个开销与本节先前给出的下界不一致，不过这只是表面上的不一致。

给定平面上 n 个点的集合 S，过 S 中各点画一条水平线和一条垂线。这些线划分平面为 $(n+1)^2$ 个矩形。另外，给定范围 $D=[x_1,x_2]\times[y_1,y_2]$，点 $p_1(x_1,y_1)\in$ 矩形 C_1 以及点 $p_2(x_2,y_2)\in$ 矩形 C_2。若在 C_1、C_2 中移动 p_1 或 p_2，则存取集合（即 S'）不变。也就是说，矩形对构成关于查找范围的等价类。所以必须考虑的不同范围的个数的上界为 $\binom{n+1}{2}\times\binom{n+1}{2}=O(n^4)$。如果预先计算每个矩形对的检索集（耗费 $O(n^5)$ 存储），则执行一次存取便可完成查找。为此，一个任意范围 D 必须要映射到一个矩形对，即一个任意点要映射到一个矩形。利用 S 点集的 x 坐标和 y 坐标集合上的对分查找能完成这一点。对分查找耗费 $O(\log n)$ 时间，再加上一次存取的 $O(1)$ 时间，便解决了上面的表面上不一致的问题。总之，该方法的耗费为 $(n^5,\log n)$。

为了减少上述方法（即矩形方法）存储耗费的巨大开销，一种改进的方法是把一维查找和直接存取范围定位结合起来，即在指定 S 点集的横坐标对 (x',x'') 之间的点集上进行纵坐标的二叉查找，用 $O(\log n)$ 时间把一个 x 范围 $(x'\leqslant x\leqslant x'')$ 变换为 $[1,n+1]$ 中的一个

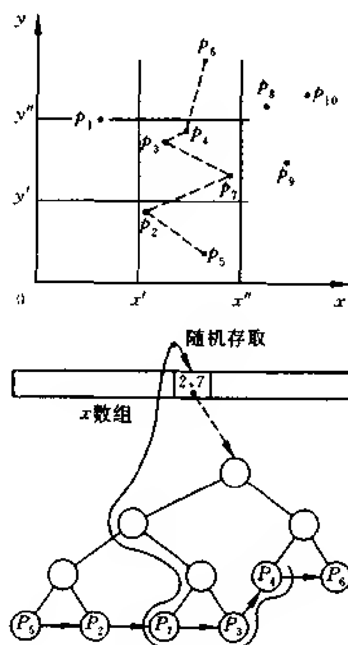


图 1-15　直接存取范围查找

整数对$(i,j)(i<j)$。该整数对与直接查找数组中的一个地址对应,它包含一个指示字指向具有$(j-i)$个叶子的二叉树,如图1-15所示。这种方法的存储为$O(n^3)$,而查找时间仍为$O(\log n)$。

另外,Bentley和Maurer提出了一个多阶段方法,以2阶段为例,其思想是连续地使用一个粗标准规模和一个细的标准规模,一个任意的范围至多被分割成为三个区间,其中粗的标准规模一个,细的标准规模两个,如图1-16所示。每个标准规模对应于一个直接查找数组。这种方法的耗费为$(n^2,\log n)$。

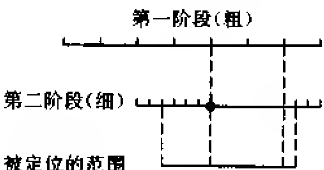


图1-16 2阶段的直接存取方法的说明

1.2.3 范围树方法

直接存取方法之所以有较高的查找时间效率,是因为构造了一批标准区间。本节介绍的方法是尽可能减少标准区间的数目,从而进一步提高查找时间的效率,这就是范围树方法的基本思想。

在x轴上给定n个点的集合$\{x_1,\cdots,x_n\}$,这n个点将x轴划分为$n-1$个区间$[i,i+1]$,$(i=\overline{1,n-1})$。利用线段树$T(1,n)$将区间$[x_1,x_n]$划分为$2\lceil\log n\rceil-2$个标准区间,每个标准区间关联于$T(1,n)$的一个结点,且确定$[i,j]$的划分的结点称为$[i,j]$的分配结点。

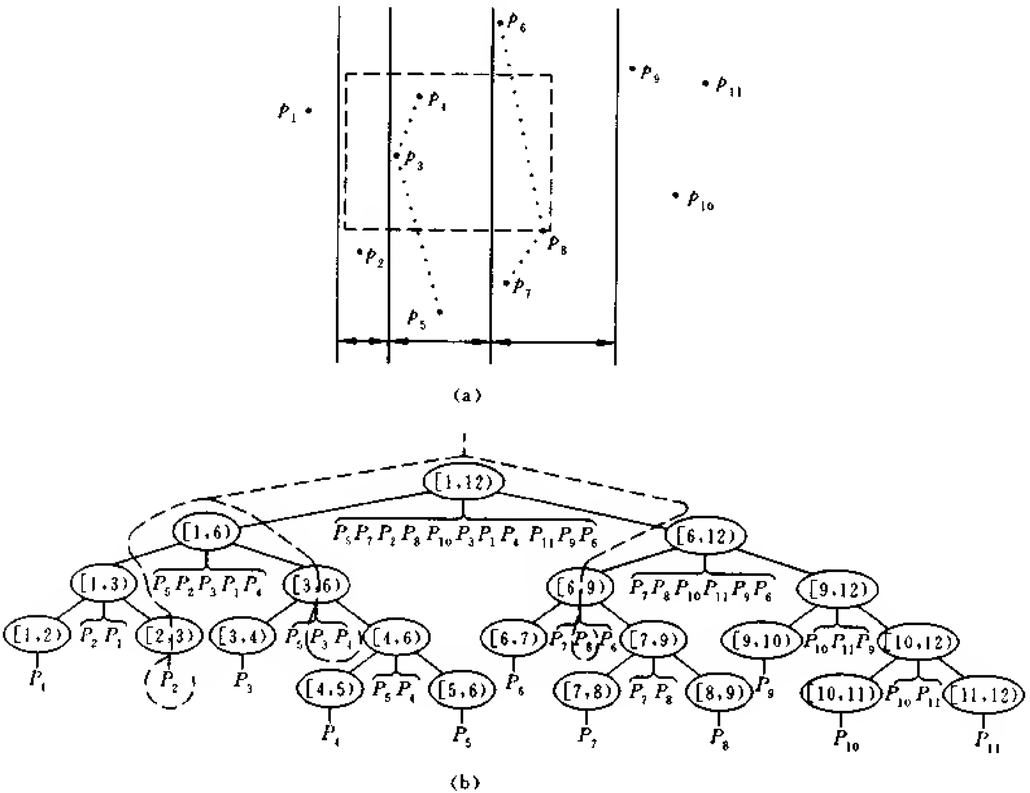


图1-17 范围树方法的说明

完成此项工作后，在查找范围 D 中便可以使用线段树。以二维情况为例，查找 x 坐标使用线段树 $T(1,n)$，该查找确定一个唯一的结点集(分配结点)。每个结点 v 对应于 $E[v]-B[v]$个横坐标的集合，其中 $E[v]$,$B[v]$分别表示 v 所代表区间的末、始端点。同样处理这些点的 y 坐标，即组织成 y 方向范围查找的标准二叉树。这就构造了范围树的数据结构，如图 1-17 所示。图 1-17(a)表示 x 方向的标准区间的划分，范围 D(虚线表示的矩形)的 x 轴方向被划分成三个标准区间，图 1-17(b)表示查找操作。图 1-17 所示范围树的基本结构从属于点集 S 的横坐标的线段树，该树的每个结点有一个指示字指向线索二叉树。

范围树方法可以推广到 $d(d>2)$维，其细节就不阐述了。在该方法中，线段树起重要的作用，它不仅是把区间分割为对数段数的一种工具，而且还为用分治法解决查找范围的划分提供了一种理想的途径。

可以证明，$d=2$ 时，范围树方法耗费询问时间 $O((\log n)^2+k)$、存储 $O(n\log n)$和预处理时间 $O(n\log n)$可以完成 n 个点集的范围查找。虽然查找时间稍有增加，但存储耗费已减少许多。

另外，利用分层范围树方法可以更好地解决范围查找问题，其预处理时间、存储和查找时间分别为 $O(n\log n)$、$O(n\log n)$和 $O(\log n)$。

1.3 判定点集是否在多边形内

上一节讨论了询问域 D 是矩形域时，报告 D 内包含点集 S 中哪些点的一些算法。本节将询问域 D 扩展为简单多边形 P，判定点集 S 中哪些点落入多边形 P 内。下面介绍周培德提出的一个算法，该算法不需判断 S 中的每个点是否在多边形 P 的内部。其基本思想是，反复求剩余点集的凸壳 C，只要判断凸壳 C 的顶点是否在 P 内，及 P 的顶点(m 个点)是否在 C 外，便可确定 S 中哪些点位于多边形 P 的内部。当平面上 n 个点呈均匀独立随机分布时，其凸壳顶点数 c 与点集的点数 n 满足关系式：

$$c=O\left(n^{\frac{1}{3}}\right)$$

因此，当第一次求得的凸壳顶点全部位于 P 内，并且 P 的全部顶点位于 C 外时，只要耗费 $O(n\log n)+O\left(mn^{\frac{1}{3}}\right)+O\left(n^{\frac{1}{3}}m\right)=O(n\log n)+O\left(nm^{\frac{1}{3}}\right)$次比较便可确定 n 个点全部落入 P 内，不必判断凸壳 C 内的 S 中的点是否在 P 内。

$Z_{1\text{-}3}$算法(判定点集是否在多边形内部的算法)

输入 n 个点$(q_1,q_2,\cdots,q_n)$的点集 S。任意多边形 P，其顶点序列为 $p_1,p_2,\cdots,p_m$。

输出 点 $q_1,q_2,\cdots,q_k(k\leqslant n)$位于 P 内。

步 1 $i\leftarrow 1,S_i\leftarrow S,C_i'\leftarrow\varnothing,C_i''\leftarrow\varnothing,C_0'\leftarrow\varnothing,C_0''\leftarrow\varnothing$。

步 2 求点集 S_i 的凸壳 C_i，设 $C_i=\{q_{i1},q_{i2},\cdots,q_{in}\}$。

步 3 **if** P 的所有顶点在 C_i 的外部 $\wedge C_i$ 的所有顶点在 P 内

then 输出"$C_1',C_2',\cdots,C_{i-1}',S_i$ 在 P 内"，终止

else **if** P 的所有顶点在 C_i 的外部 $\wedge C_i$ 的所有顶点在 P 的外部 $\wedge P$ 的边与 C_i 的边不相交

then　输出"$C_1'',C_2'',\cdots,C_{i-1}'',S_i$ 在 P 外"，终止

else　goto　步 4

步 4　设 $C_i'=\{q_{i1},q_{i2},\cdots,q_{ij}\}$ 在 P 内，$0\leqslant j\leqslant i_1$，$j=0$ 时，$C_i'=\varnothing$。保留 C_i'，$C_i''=C_i-C_i'$。

步 5　$S_{i+1}\leftarrow S_i-C_i$，$i\leftarrow i+1$，重复步 2，3，4 直到 S_i 为空集。

步 6　输出"$C_1',C_2',\cdots,C_{i-1}'$ 在 P 内"，终止。

定理 1-3　给定平面域 $E=[0,A]^2$ 中 m 个顶点 $(p_1,p_2,\cdots,p_m)$ 的多边形 P，设点集 $S=\{q_1,q_2,\cdots,q_n\}$ 均匀分布于 E 中，则算法 $Z_{1\text{-}3}$ 正确地判断了 S 中哪些点位于 P 的内部或外部，而且算法在最坏情况下的复杂性为 $\max[O(mn),O(ln\log n)]$ 次比较和 $O(ln)$ 次乘法，其中 l 是点集 S 的凸壳层数。

证明　平面点集 S 与多边形 P 的位置关系只有下述三种情况：多边形 P 位于点集 S 之中；点集 S 位于多边形 P 内；点集 S 中一部分点处于 P 内。如果先求出点集 S 的凸壳 C，则上述关系可以表示为：C 包含 P；P 包含 C；C 与 P 相交。因此在第一、二种情况下，只要确定 C 的顶点与 P 的顶点的位置关系便可正确地判定 S 中点的位置。对于第三种情况，即 S 中有一部分点位于 P 内，另一部分点位于 P 外，是采用逐次求凸壳 C_i，并判断 C_i 中各顶点的位置，保留位于 P 内的 C_i 顶点（记为 C_i'），求出位于 P 外的 C_i 顶点（记为 C_i''），直到发生第一、二种情况或剩余点集 S_i 为空。这样，每次求得的 C_i' 均处于 P 内，而 C_i'' 位于 P 外，$Z_{1\text{-}3}$ 算法是上述方法的实现步骤，因此该算法正确地判断了 S 中的各点的位置（位于 P 的内部或外部）。

算法的步 1 和步 6 只需常数时间。步 2 耗费 $O(n\log n)$ 次比较和 $O(n)$ 次乘法可以求得点集 S 的凸壳 C_1。判定 1 个点是否在 P 内，用 $O(m)$ 次比较；判定 1 个点是否在 C_1 内，用 $O(i_1)$ 次比较。假定步 2，3，4 与步 5 的循环次数为 l，即要求 l 次凸壳：$C_1,C_2,\cdots,C_l$，每层凸壳所含顶点数相同，即 $\frac{n}{l}$ 个顶点，则一次执行步 3 需要 $O\left(\frac{n}{l}\cdot m\right)+O\left(m\cdot\frac{n}{l}\right)$ 次比较，步 3 循环执行 l 次，需要 $O(m\cdot n)$ 次比较（S_i 为空时，步 3 才执行 l 次；否则，步 3 不必执行 l 次）。步 4 耗费 $\frac{n}{l}$ 次比较可以求得 C_i''。步 5 用 n 次比较求得 S_{i+1}。因此算法需要的比较次数为

$$
\begin{aligned}
O(n\log n)&+O\left(\left(n-\frac{n}{l}\right)\log\left(n-\frac{n}{l}\right)\right)+\cdots+O\left(\left(n-\frac{(l-1)n}{l}\right)\log\left(n-\frac{(n-1)n}{l}\right)\right)\\
&+O(m\cdot n)+n+\left(n+\left(n-\frac{n}{l}\right)+\cdots+\frac{n}{l}\right)\leqslant O(ln\log n)+O(m\cdot n)\\
&=\max(O(ln\log n),O(m\cdot n))
\end{aligned}
$$

乘法次数为

$$
\begin{aligned}
O(n)+O\left(n-\frac{n}{l}\right)+\cdots+O\left(n-\frac{(n-1)n}{l}\right)&=O\left(\left(2n-n-\frac{n}{l}\right)\frac{1}{2}\right)\\
&=O\left(\frac{ln}{2}-\frac{n}{2}\right)=O(ln)
\end{aligned}
$$

证毕。

上述求得的复杂性是最坏情况下的复杂性。在大多数情况下，$Z_{1\text{-}3}$算法比逐点判定的方法要好得多。因为当 $S_i \neq \varnothing$，并且算法于步 3 终止时，S_i 内的点不必判定，便可确定是在 P 内或 P 外，特别是当点集 S 中的点服从正态分布，并且密度较高的部分位于 P 内或 P 外时，该算法将获得较高效益。最坏情况下，算法 $Z_{1\text{-}3}$所需要的乘法次数 $O(ln)$比逐点判断所需要的乘法次数 $O(mn)$少得多。

1.4 平面中线段集和空间中三角形集的正交询问

1.2 节中介绍的分层范围树方法耗费 $O(n\log n)$存储和 $O(\log n)$查找时间解决了范围查找问题，后来对于 $d=2$，Chazelle 把空间耗费减少到 $O(n\log n)/O(\log\log n)$。正交域查找的空间-时间折衷是一个重要课题。对于非点的几何对象，比如平面上正交线段集中的正交询问已经知道几个结果(Chazelle，1986；Edahiro，1987)。另外，对象和窗口是非正交的情况下，Dobkin 和 Edelsbrunner 已进行了研究，他们是把这种情况作为空间细分以及单纯形域查找的应用来研究的。

本节讨论非正交对象中的正交询问，特别是 d 维空间中的线段和三角形，如图 1-18 所示。为了区分非点对象集中的正交询问与点集中的正交询问，有时称它为正交限制。正交限制问题描述如下：平面上给定 n 条线段(可能相交)的集合 L 及矩形域 D(又称窗口)，询问 L 中哪些线段穿过 D 的边而进入 D 内。对于三维空间可以提出类似问题，只是把 L 改为三角形集并且 D 改为长方体域。

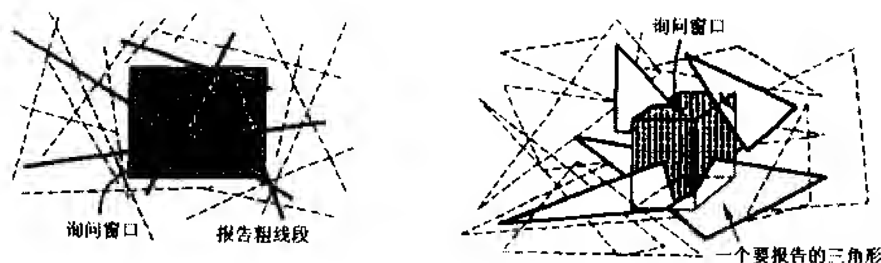


图 1-18 平面上的线段和空间中的三角形的正交限制

下面介绍关于正交限制的一些方法。对于报告询问问题，一种方法如下：分割每个对象成小块，使每个小块近似于一个点，然后把正交询问应用于有关的点集。另一种方法是 Edahiro 提出的吊桶法。上述方法的有效性取决于输入的离散性及解析性质(比如，长度、面积、分布等)，该解析性使得对方法的分析变得困难了，因此希望设计出的方法仅取决于输入的几何性质的某些离散参数。

给定平面上 n 条线段，设 K 是这些线段排列的复杂性(即这些线段的端点和交点的数目)。如果有 $O(K+n\log n)$空间可供使用，那么利用平面划分和相关的树结构可以构造出一个 $O(\log n)$时间的询问结构。但 $O(K)$空间太大了，要研究空间-询问折衷以便在给定限界的存储中存储数据。

给定 d 维空间中 n 条线段的集合，定义 K 为把线段投影到二维空间所得到的排列复

杂性的和。显然，$K=O(n^2)$。为简单起见，用符号 $\widetilde{O}$ 表示在多对数因子范围内的时间复杂性。下面既考虑计数询问又考虑报告询问。

定理 1-4 对于任意 $m \geqslant n\log^{d-1}n$，可以构造出 d 维空间中 n 条线段的正交计数询问的 $O(m)$ 空间的数据结构，并耗费 $\widetilde{O}(\sqrt{K/m})$ 询问时间。而报告询问需要与输出规模成比例的附加时间。

为了报告与正交询问窗口相交的线段，只要报告与窗口的每个小面相交的线段，这些线段必有一个端点在窗口中。因此，通过使用正交域查找可以找到。考虑垂直于第 d 条坐标轴的小面，如果把第 d 条坐标轴作为时间参数，那么线段集中的线段可以视为线性移动点的集合。如果考虑 $d-1$ 维正交域查找的那些点的域树，那么组合变化数是 $\widetilde{O}(K+n)$。因此，如果 $m=K$，定理 1-4 意味着正交域查找的对数询问查找结构。该定理给出了结构的空间-询问折衷。

类似地，可以求解 d 维情况下三角形集合中的正交限制。并且在 $\widetilde{O}(\sqrt{K/n}+\sqrt{M}/\sqrt[3]{n})$ 时间内查询到距询问点最近的三角形。

1.4.1 吊床询问及推广的吊床询问

本小节介绍吊床(Hammock)询问及推广的吊床询问的概念并阐述几个结果。考虑具有垂直坐标系统$(x_1,x_2,\cdots,x_d)$的 d 维空间。设 H 是 d 维空间 E^d 中维数同为$(d-K)$的 n 个仿射子空间的排列。给定具有常数面的 K 维凸域 Q，定义管 $Z(Q)=\{x\in E^d \mid (x_1,\cdots,x_k)\in Q\}$，称 $\mathcal{Z}(Q)=Z(Q)\cap A(H)$为推广的吊床。如果 $k=d-1$，称推广的吊床为吊床，因此，$A(H)$是关于吊床的超平面的排列。

吊床询问 给定位于吊床 $Z(Q)$中的一条询问线段，计数(或报告)与它相交的超平面，如图 1-19 所示。

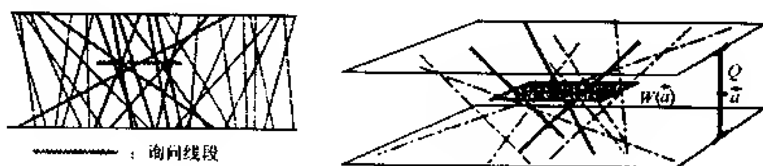


图 1-19 吊床询问和推广的吊床询问($d=3,k=1$)

推广的吊床询问 设 S 是 d 维空间中的 $k(k\leqslant d-1)$维仿射子空间的集合。令 Q 是 k 维凸集，并且 $\mathcal{Z}(Q)$是 E^d 中相关的推广吊床。给定一个$(d-k)$维正交窗口 W 及一个 k 维矢量 $a\in Q$，定义 $Z(Q)$中 W 的平移复制窗口 $W(a)=a\oplus W=\{x\in E^d \mid (x_1,\cdots,x_k)=a$ *and* $(x_{k+1},\cdots,x_d)\in W\}$。询问与询问窗口 $W(a)$相交的 S 的元素集合。

下面先考虑吊床询问，然后利用吊床询问求解推广的吊床询问。把推广的吊床询问应用于线段和三角形的正交限制时，k 为 1 和 2。

设 M 是 n 与相交于吊床内超平面对数目之和，显然，$M=O(n^2)$。

在对偶条件下，将 n 个超平面变换成对偶空间中的 n 个点。因此，吊床询问的对偶是

单纯形域查找的一种特殊情况。更确切地说，吊床询问的对偶是与超平面对的楔形有关的域查找，其中超平面的法矢量在 Q 中。Matoušek 给出了下述结果。

定理 1-5 存在一个 $O(n)$ 空间和 $O(n^{1-1/d})$ 询问时间域查找数据结构。对于任意正常数 δ 预处理时间是 $O(n^{1+\delta})$。对于 $n<m<n^2$，存在一个 $O(m)$ 空间和 $O\left(\frac{n}{m^{1/d}}\log^3 n\right)$ 询问时间域查找数据结构，该数据结构在预处理时间 $O(n^{1+\delta}+m(\log n)^{\delta})$ 内被构造出来。如果域是楔形，则询问时间为 $O\left(\frac{n}{m^{1/d}}\log^{1.5} n\right)$。

更进一步，Takeshi Tokuyama 得出下述结果。

定理 1-6 对于 $m\geqslant n$，存在一个 $O(m)$ 空间的数据结构，使得管中的询问线段所交的超平面数可以在 $O\left(\left(1+\frac{\sqrt{M}}{m^{1/d}}\right)\log^{2+1/d} n\right)$ 时间内计算，在与输出规模成比例的附加时间内可以报告超平面，对于 $d\leqslant 3$，预处理时间为 $O(n^{1+\epsilon})+\widetilde{O}(m)$，而 $d\geqslant 4$ 时，预处理时间为 $O(n^2)+\widetilde{O}(m)$。

证明略。

下面考虑推广的吊床询问。如果 $k=d-1$，则推广的吊床询问是吊床询问的一种特殊情况，这里询问线段总是垂直于 Q。

对于 $k+1\leqslant i\leqslant d$，设 $S(i)$ 是通过把 S 投影到由 $x_1,\cdots,x_k$ 及 x_i 坐标所生成的 $k+1$ 维子空间得到的超平面集。易见，$S(i)$ 成为 $k+1$ 维空间中的吊床，定义 $M_i(k+1\leqslant i\leqslant d)$ 为 Q 和 x_i 坐标的直积(direct product)的管空间内 $S(i)$ 的超平面相交对的数目。设 $M=\sum_{i=k+1}^{d} M_i$。

定理 1-7 如果有 $O(m)$ 空间可供使用，其中 $m\geqslant n\log^{d-k-1} n$，那么可以找到有 $O\left(\left(1+\frac{M^{1/2}}{m^{1/(k+1)}}\right)\log^{r(d)} n\right)$ 询问时间的推广吊床询问数据结构，其中 $r(d)=2(d-k)+2(d-k)/(k+1)$。$d\leqslant 3$ 时，预处理时间为 $O(n^{1+\epsilon})+\widetilde{O}(m)$，而 $d\geqslant 4$ 时，为 $\widetilde{O}(m+n^2)$。

1.4.2 正交限制

本小节将不加证明地介绍有关正交限制的一些结论。

1. 线段的正交限制

d 维空间 E^d 中给定 n 条线段的集合 L，预处理 L，使得能有效地计数(或报告)与平行于坐标轴的窗口 W 相交的线段集。

设 N 是与 W 相交的线段数，N_1 是 W 中端点数，N_2 为线段与 ∂W(即 W 的边界)之间的交点数。如果一条线段与 ∂W 相交两次，则计数两个交点。因此，$N=\frac{1}{2}(N_1+N_2)$。由正交询问可以计算 N_1。

这样，可以把线段正交限制归约为计数(或报告)穿过每个小面的线段的问题。在 E^d 的笛卡尔坐标系 $(x_1,\cdots,x_{d-1},x_d)$ 中令 $x_d=z$，如果垂直于 z 轴的超平面是水平的，则计数

(或报告)与询问窗口 W 的水平面 F 相交的线段。设 $K_{i,j}$表示线段投影到由 x_i 和 x_j 确定的二维子空间所得到的排列复杂性，定义 $K=\sum_{i<j} K_{i,j}$。

把 L 中的线段投影到 z 轴得到区间集合 I，令 $P_1,\cdots,P_{2n}$是这些区间的端点，将端点集合构成区间树。把 I 中区间存储在区间树中便产生了线段树。

定理 1-8 d 维空间中给定 n 条线段，对于任意 $m\geqslant n\log^{d-1}n$，可以构造 $O(m)$空间的数据结构，并且在 $O\left(\left(1+\frac{K}{m}\right)^{1/2}\log^{(5d-3)/2}n\right)$时间内完成正交限制询问。

证明略。

2. 三角形的正交限制

d 维空间 E^d 中给定 n 个三角形的集合 S，预处理 S，使得能有效地报告与窗口 W 相交的三角形集合，并且 W 的棱相互正交，且平行于坐标轴。

给定询问窗口 W，如果(1)三角形 T 的一个顶点位于 W 内；(2)T 的一条边与 W 的一个面相交；(3)T 与 W 的$(d-2)$维面(称为棱)之一相交，则 T 与 W 相交。利用线段的正交限制可以处理情况(1)和(2)。下面只讨论情况(3)。

设 F 是 W 的棱，并且 F 垂直于由 x_1 和 x_2 生成的二维空间 H，F 垂直投影到 H 上的象点记为 p。如果 F 与 T 相交，则 T 到 H 的投影包含 p。但反之不成立。设 $aff(T)$是包含 T 的二维仿射空间，则有下述引理。

引理 1-1 T 与 F 相交，当且仅当 T 投影到 H 上的象包含 p 并且 $aff(T)$与 F 相交。

由引理 1-1，问题归约为推广的吊床询问$(k=2)$。再由定理 1-7，有下述定理。

定理 1-9 存在 $O(m)$空间的数据结构$(m\geqslant n\log^{d-1}n)$，使得在 $\widetilde{O}\left((K/m)^{\frac{1}{2}}+M^{\frac{1}{2}}/m^{\frac{1}{3}}\right)$时间内完成 n 个三角形中的正交限制，其中(1)M 是彼此相交的三角形对的数目，如果把它们投影到轴的三维空间；(2)如果把它们投影到轴的二维空间，则 K 是彼此相交的三角形边对的数目。

定理 1-10 存在 $O(m)$空间的数据结构$(m\geqslant n\log^{d-1}n)$，使得在 $\widetilde{O}\left((K/m)^{\frac{1}{2}}+M^{\frac{1}{2}}/m^{\frac{1}{3}}\right)$时间内找到距询问点最近的三角形。

第2章 多 边 形

多边形分为凸多边形与任意多边形。这里所说的多边形是指简单多边形，即边之间除顶点外不相交的多边形，它隐含多边形的每个顶点的度数为2这一条件。凸多边形是一种特殊的多边形。本章将首先介绍凸多边形，然后介绍任意多边形(简称多边形)以及多边形的三角剖分和凸划分(即划分成凸多边形)。

2.1 凸多边形

凸多边形是一种特殊的多边形，它具有许多特性，比如凸多边形的所有顶点均在任一条边的同一侧等，因此在许多问题中常常将多边形分割成凸多边形，特别是分割成三角形。本节介绍与凸多边形有关的问题的算法，包括判定点是否在凸多边形的内部、确定线与凸多边形的交、判定两个凸多边形是否相交以及确定两个凸多边形的交。

定义 2-1 由平面上若干条线段$\overline{p_1p_2}, \overline{p_2p_3}, \cdots, \overline{p_np_1}$围成的封闭有界域称为多边形。其中线段$\overline{p_ip_{i+1}}$($i=\overline{1,n}, p_{n+1}=p_1$)称为多边形的边，相邻的两条边仅在端点相交，其交点p_i称为多边形的顶点。

多边形的所有边组成多边形的边界，多边形的边界将平面划分成内部和外部两部分，其中内部是有界的，而外部是无界的。这里定义多边形为平面的一个封闭有界域，但通常认为多边形是指多边形边界而不是域本身。下面用∂p表示多边形P的边界。

定义 2-2 多边形中具有共同端点的边称为相邻的边，比如$\overline{p_{i-1}p_i}$与$\overline{p_ip_{i+1}}$是两条相邻的边。若多边形中不相邻的边不相交，则称该多边形为简单多边形。

定义 2-3 与同一顶点p_i关联的两条边$\overline{p_{i-1}p_i}$，$\overline{p_ip_{i+1}}$形成的位于多边形内部的角$\angle p_{i-1}p_ip_{i+1}$称为多边形的内角。

定义 2-4 若多边形P的顶点序列$p_1, p_2, \cdots, p_n$按逆时针方向排列，并且点p_{i+1}在$\overrightarrow{p_{i-1}p_i}$的左(右)侧，则称点$p_i$为凸(凹)点。以凸(凹)点为顶点的内角$\leqslant\pi$($>\pi$)。

定义 2-5 所有内角都小于π的多边形称为凸多边形。凸多边形的所有顶点均为凸点。

先求出凸多边形各顶点y坐标最小值所对应的顶点，设为p_1。然后，如果p_{i+1}在$\overrightarrow{p_{i-1}p_i}$的左(右)侧($i=\overline{2,n}$)，那么凸多边形顶点序列是按逆(顺)时针方向排列。对于任意简单多边形，可以先求出多边形顶点的凸壳，然后用上述方法确定多边形顶点序列是逆时针排列还是顺时针排列。

凸多边形或多边形用其顶点序列$p_1, p_2, \cdots, p_n$(逆时针方向排列)来表示。另外，还可以用均衡分层树的结构表示凸多边形，在这种表示下可以得到一些重要结果。

定义 2-6 如果(1)P_0是至多有4个顶点的多边形；(2)$P_k=P$；(3) 删去P_i的某些顶

点可以得到 P_{i-1}，则多边形序列 $P_0,P_1,\cdots,P_k$ 是凸多边形 P 的均衡分层表示。

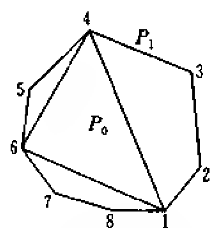


图 2-1 凸多边形的均衡分层表示

图 2-1 是有 8 个顶点凸多边形的均衡分层表示，多边形 P_1 由 8 个顶点组成，从 P_1 中删去顶点 2,3,5,7,8 得到多边形 P_0，即 P_0 由顶点 1,4,6 组成。

如果把凸多边形边的序列存储在均衡树的叶中，即一个叶存储一条边，那么树的每一级对应分层表示中的一个多边形，如图2-2所示。图中根结点是多边形 $P_0=\{1,5\}$，深度为 1 的结点表示多边形 $P_1=\{1,3,5,7,1\}$，即根结点的左子结点表示用边链(1,3)，(3,5)代替 P_0 的边(1,5)，右子结点表示用边链(5,7)，(7,1)代替 P_0 的边(5,1)。依此类推。均衡树中所有内结点(叶除外)所表示的多边形恰好是凸多边形 P 的一种划分。

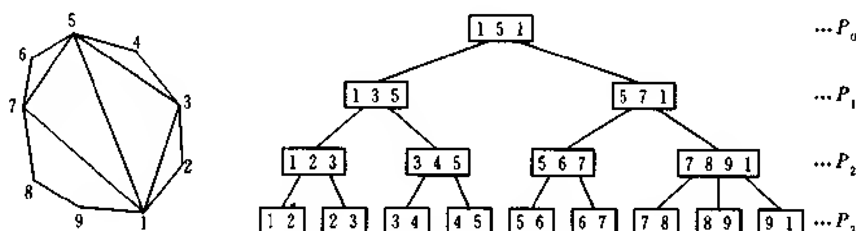


图 2-2 凸多边形与均衡分层树

利用上述凸多边形的均衡分层表示可以证明下述结论。

引理 2-1 (1) 在时间 $O(n)$ 内可以构造凸多边形的均衡分层表示，其中 n 是凸多边形 P 的顶点数；(2) 如果 $P_0,P_1,\cdots,P_k$ 是 P 的均衡分层表示，那么 $k=O(\log n)$。

证明 设凸多边形 P 的顶点数为 n，并设处理一个顶点耗费一个单位时间，由凸多边形的均衡分层表示的定义知，每个顶点只处理一次，共计 n 个顶点，所以时间为 $O(n)$。

在均衡分层树中，第 i 层结点表示 P_i，共计 $\log n$ 层，所以 $k=\log n=O(\log n)$。

定理 2-1 给定凸多边形 P 的均衡分层表示及 P 的内点 q 和任意点 x，在时间 $O(\log n)$ 内可以确定 x 是否在 P 内，其中 n 是 P 的顶点数。

证明 从 q 向 P 的各顶点引射线，得到 n 个扇形，这些扇形将平面分成 n 个域，按逆时针方向编码 n 个扇形。设 T 是 P 的均衡分层表示树，该树的根结点存储的顶点和 q 至多构成 4 个扇形，耗费 $O(1)$ 时间可以检查哪个扇形包含 x。对包含 x 的扇形继续检查根的某个子结点存储的顶点和 q 构成的扇形，这些扇形将包含 x 的原扇形至多细分成 4 个部分，再耗费 $O(1)$ 时间可以检查哪个小扇形包含 x。对包含 x 的小扇形继续该过程。由于均衡分层树的深度为 $\log n$，故此检查过程重复 $\log n$ 次，可以确定 x 落入哪个基本扇形(该基本扇形不可能再细分)，此时终止该过程。该过程循环 $\log n$ 次，每次循环耗费 $O(1)$ 时间，所以时间耗费为 $O(\log n)$。最后再用 $O(1)$ 时间判定 x 是否位于基本扇形与 P 的交(该交是一个三角形)。 证毕。

定理 2-2 给定凸 n 边形 P 的均衡分层表示和直线 L，在 $O(\log n)$ 时间内可以确定 L

与 P 是否相交并确定 L 与 P 的交。

证明 取 P 的不相邻两顶点连线的中点作为 P 的内点 q，然后采用定理 2-1 证明中的方法作扇形，仍设 T 为 P 的均衡分层表示树。从根结点开始，检查结点存储的各顶点位于 L 的哪一侧，如果有两个相邻的顶点分别位于 L 的两侧，则 L 与 P 相交；否则，检查根的某个子结点存储的顶点位于 L 的哪一侧，依此类推。在 T 中沿着从根到叶的某条路径至多检查 $\log n$ 个结点，检查每个结点所存储的顶点位于 L 的哪一侧需要 $O(1)$ 时间，因此至多耗费 $O(\log n)$ 时间可以确定 L 与 P 是否相交及确定 L 与哪些 P 边相交，并计算 L 与 P 的交（位于 P 内的一线段）。 证毕。

定理 2-3 给定凸 n 边形 P 的均衡分层表示和(1)线段 S，则在时间 $O(\log n)$ 内可以计算 $P\cap S$；(2)点 p，则在时间 $O(\log n)$ 内可以判定 p 是否在 P 内。

证明 (1) 设线段 S 是直线 L 的一段，由定理 2-2，在时间 $O(\log n)$ 内可以计算 $P\cap L$，$P\cap L$ 是一条线段。由 $P\cap L$，在常数时间内可以计算 $(P\cap L)\cap S=P\cap S$。

(2) 可以把点 p 看成是线段 S 的退化，由(1)即知结论成立。 证毕。

如果 $p_1,p_2,\cdots,p_k$ 是一多边形顶点序列，并且 $y(p_i)>y(p_{i+1})(i=\overline{1,k-1})$，则称 p_1，$p_2,\cdots,p_k$ 是单调多边形链。如果对此链增加两条半无穷水平线 $y=y(p_1)$ 和 $y=y(p_k)$，那么这就定义了一无界凸域，如图 2-3 所示。给定凸 n 边形 P 的均衡分层表示，在 $O(\log n)$ 时间内可以分裂 P 成两个单调多边形链 P_L 和 P_R，其中 $P_L(P_R)$ 是左(右)封闭的，如图2-4所示。



图 2-3 无界凸域

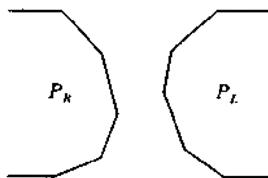


图 2-4 P_R 与 P_L

引理 2-2 设 P,Q 是两个凸多边形，则 $P\cap Q\neq\varnothing$ iff $P_L\cap Q_R\neq\varnothing$ and $P_R\cap Q_L\neq\varnothing$。

证明 由于 $P=P_L\cap P_R$ 和 $Q=Q_L\cap Q_R$，则有 $\varnothing\neq P\cap Q=P_L\cap P_R\cap Q_L\cap Q_R$，即 $P_L\cap Q_R\neq\varnothing$ 与 $P_R\cap Q_L\neq\varnothing$。

如果 $P\cap Q=\varnothing$，则存在分隔 P 和 Q 的直线 L，若 L 是水平直线，那么显然有 $P_L\cap Q_R=P_R\cap Q_L=\varnothing$；若 L 不是水平线，不失一般性，设 P 位于 L 的左侧，则 $P_R\cap Q_L=\varnothing$。

证毕。

引理 2-2 表明，判定两个凸多边形是否相交的问题转换为判定两条单调多边形链是否相交。设 $R(L)$ 是一右(左)闭的单调多边形链，$r_1,r_2,\cdots,r_m(l_1,l_2,\cdots,l_n)$ 是 $R(L)$ 的边，其中 r_1,r_m,l_1,l_n 是 4 条水平射线并且 $r_1(y)>r_m(y)$，$l_1(y)>l_n(y)$，而其他边都是有限的。利用一种变形的二叉搜索方法可以确定 R 和 L 是否相交。设 $i=\lfloor(m+1)/2\rfloor$，$j=\lfloor(n+1)/2\rfloor$ 及 $R_i(L_j)$ 是与线段 $r_i(l_j)$ 共线的直线($R_i(L_j)$ 称为 $r_i(l_j)$ 的支撑线)，若 R_i 与 L_j 相交(否则 L_{j+1} 将与 R_i 相交，因为假设相邻边不共线)，则 R_i 与 L_j 划分平面成 4 个域，这些域

分别记为 $L'R'$,L'、R'和空。R 和 L 可能分别位于其中两个域中,它们也许位于同一个域中,如图 2-5 所示。链 R 可以在域 R'和 $L'R'$中。由 R 和 L 定义 4 条新的单调链如下:R_{top}由边 $r_1,r_2,\cdots,r_i$ 和终止于 r_i 下端点的水平射线组成,而 R_{bot}则由边 $r_i,r_{i+1},\cdots,r_m$ 和终止于 r_i 上端点的水平射线组成,类似定义 L_{top}和 L_{bot}。

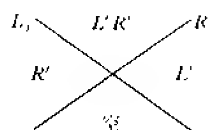


图 2-5 R_i 与 L_j 划分平面成 4 个域

引理 2-3 如果直线 R_i 和 L_j 相交而线段 r_i 和 l_j 不相交,又设域 $L'R'$在空域的上方,则(1) 如果 r_i 的下端点不在 $L'R'$ 域中,则 $R \cap L \neq \varnothing$ iff $R_{top} \cap L \neq \varnothing$;(2) 如果 l_j 的下端点不在 $L'R'$ 域中,则 $R \cap L \neq \varnothing$ iff $R \cap L_{top} \neq \varnothing$;(3) 如果 r_i 和 l_j 的两个端点都在 $L'R'$域中,r_i 的下端点的 y 坐标不大(小)于 l_j 的下端点的 y 坐标,则 $R \cap L \neq \varnothing$ iff $R_{top} \cap L \neq \varnothing(R \cap L_{top} \neq \varnothing)$。

证明 (1) 如果 r_i 的下端点不在 $L'R'$域中,则 R 的边 $r_{i+1},\cdots,r_m$ 全在域 R'中,因此不可能与链 L 相交,而且新的边或 R_{top}不与 L 相交,所以 $R \cap L \neq \varnothing$ iff $R_{top} \cap L \neq \varnothing$。

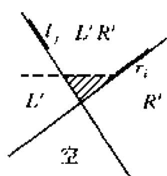


图 2-6 引理 2-3 (3)证明的示意图

证明(2)与(1)类似。

证明(3),设 r_i 和 l_j 的端点都位于 $L'R'$ 域中并且 r_i 的下端点的 y 坐标不大于 l_j 的下端点的 y 坐标。为了证明 $R_{top} \cap L \neq \varnothing$ iff $R \cap L \neq \varnothing$,首先设 $R_{top} \cap L = \varnothing$,但 $R \cap L \neq \varnothing$,则边 $r_{i+1},\cdots,r_m$ 中必有一条边与 L 相交,因此链 L 在图 2-6 中阴影部分必有一个点。又由于凸性链 L 必与 R_{top}相交,所以这是一个矛盾。因此 $R_{top} \cap L = \varnothing$,即 $R \cap L = \varnothing$。其次设 $R_{top} \cap L \neq \varnothing$,而 $R \cap L = \varnothing$,那么 R_{top}的唯一下水平射线(称为 r)可能与 L 相交。由 r_i 与 l_j 的相对位置,射线 r 与链 L 的上水平射线 l_1 不可能相交,因此,如果从下水平射线 l_n 开始沿链 L 检查,可以发现 $k>1$,使得 l_k 与 r 相交,继续检查链 L,便进入由链 R 定义的凸域。因为链 R 沿 x 方向趋于$-\infty$,链 L 沿 x 方向趋于$+\infty$,R 与 L 必相交,所以 $R_{top} \cap L \neq \varnothing$,即 $R \cap L \neq \varnothing$。 证毕。

引理 2-3 显示,在常数时间内多边形链(R 或 L)的边数可以减少一半。更确切地说,考虑中间边 r_i 和 l_j 以及支撑线 R_i 和 L_j,如果 R_i 和 L_j 是共线的,并且 R_i 位于 L_j 的左侧,则 L 和 R 不相交。如果 R_i 和 L_j 共线并且 R_i 位于 L_j 的右侧,则 R_{i+1}(边 r_{i+1}的支撑线)和 L_j将相交。如果 R_i 和 L_j 相交,另一情况是类似的。若线段 r_i 和 l_j 相交,则寻找交点。如果 r_i 和 l_j 不相交,由引理 2-3,可以减少链的一半,因此问题的规模也减少了。

由上述讨论可知,在 $O(\log(n+m))$时间内一条链将减少到边数目受限的链,比如至多 10,对于每条这样链的边,依据定理 2-3(1),在对数时间内可以检查它与其链的交。因此再耗费时间 $O(\log(n+m))$便可完成交的测试,这就证明了下面的定理。

定理 2-4 给定凸 n 边形 P 和凸 m 边形 Q 的均衡分层表示,则在 $O(\log(n+m))$时间内可以判定 P 与 Q 是否相交。

定理 2-5 设 P 与 Q 是凸 n 边形,则在 $O(n)$时间内可以计算 $P \cap Q$。

证明 设 P 和 Q 的顶点序列分别为 $p_1,p_2,\cdots,p_n$ 和 $q_1,q_2,\cdots,q_m$,$m \leqslant n$。令 p 是 P 内的一点,比如取 p 为顶点 $p_1,p_2,\cdots,p_n$ 的重心,在线性时间内由 P 的顶点序列可以计算出点 p。考虑始于点 p 并通过 P 的顶点的 n 条射线,这些射线划分平面为 n 个扇形。设 q_i 是

Q 的顶点，用 $S(q_j)$ 表示包含 q_j 的扇形，如图 2-7 所示。

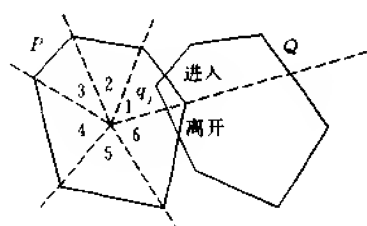


图 2-7　包含 q_j 的扇形 $S(q_j)$

首先，在 $O(n)$ 时间内可以计算 $S(q_1)$。其次，为了确定边 $\overline{q_1q_2}$ 是否与 P 的一条边相交以及 $\overline{q_1q_2}$ 是否离开扇形，只须考虑限界 $S(q_1)$ 的射线以及穿过扇形的一条 P 边。如果 $\overline{q_1q_2}$ 不离开扇形，则在 $O(1)$ 时间内可以完成；否则，相同的论证可用于已进入的扇形。因此，确定边 $\overline{q_1q_2}$ 与 P 的边的所有交需要时间 $O(1+s_1)$，其中 s_1 是穿过线段 $\overline{q_1q_2}$ 的射线数目。

类似地，计算边 $\overline{q_2q_3}$ 与 P 的边的所有交需要时间 $O(1+s_2)$，其中 s_2 是与 $\overline{q_2q_3}$ 相交的射线数目。依此类推，计算 Q 的边与 P 的边的所有交耗费 $O\left(m+\sum\limits_i s_i\right)$ 时间。又由于每条射线至多与 Q 的两条边相交，显然 $\sum\limits_i s_i \leqslant 2n$，因此，在时间 $O(n+m)$ 内计算 P 的边与 Q 的边的所有交，即计算 P 与 Q 的交。故当 $n=m$ 时，定理结论成立。

2.2　简单多边形

多边形是S计算几何研究的对象之一，这是由于多边形是许多真实物体外形的一种方便而又精确的描述工具，并且在计算上很容易实现。多边形的应用包括自动符号识别中单个符号的形状，机器人运动时要避免的障碍物的形状，或者图形屏幕上显示的固体对象的一部分形状等。然而多边形可能又是相当复杂的对象，因此经常需要把它们看成是由更简单的部分组成，这就是划分多边形的问题(下节将讨论)。本节介绍与多边形有关的某些问题。

定义 2-7　设平面上 n 个点 $p_1,p_2,\cdots,p_n$ 按循环排序方法逆时针排列，p_1 在 p_n 之后，又设 $e_1=\overline{p_1p_2},e_2=\overline{p_2p_3},\cdots,e_n=\overline{p_np_1}$ 是连接点的 n 条线段，那么这些线段界限一个多边形(简单多边形) iff (1) 循环排序中相邻线段对的交是它们之间共有的单个点：$e_i\cap e_{i+1}=p_{i+1}$；(2) 不相邻的线段不相交：$e_i\cap e_j=\varnothing,j\neq i+1;i=\overline{1,n},e_{n+1}=e_1,p_{n+1}=p_1$。

上述定义中，点 p_i 称为多边形的顶点，线段 e_i 称为多边形的边。n 个顶点的多边形必有 n 条边。图 2-8 所示多边形不是简单多边形，因为它们的边虽然满足定义 2-7 中的条件(1)，但不满足条件(2)。本书不讨论非简单多边形。

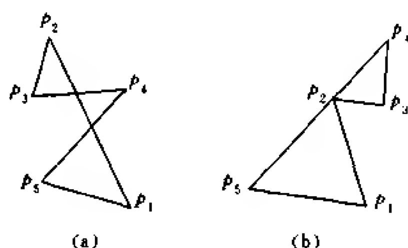


图 2-8　非简单多边形

依据定义 2-7 中规定的多边形顶点顺序及定义 2-1 的说明，如果沿着多边形边界按逆时针顺序访问顶点，那么多边形内部总是在左侧。

下面介绍几个与多边形有关的问题及其解决方法。

问题 2-1　设某建筑物的平面图如图 2-9 所示，问至少要安装多少只监视器才能监视

整座建筑物。

当然监视器不能穿透墙壁，每只监视器是从不同方向可以看见的一个固定点，它可以旋转角度 2π 监视其周围环境。图 2-9 中 4 个圆点表示 4 只监视器的位置。

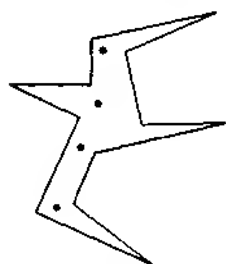


图 2-9 12 个顶点的多边形，要求 4 只监视器

设点 x，y 位于多边形 P 内，从点 x 可以看见点 y（或者从 y 可以看见 x） iff 封闭线段 $\overline{xy}$ 完全位于多边形 P 的内部：$\overline{xy}\subset P$。线段 $\overline{xy}$ 称为可视线。

一只监视器是一个点，如果多边形 P 中每个点与一组监视器中至少一只监视器所在位置的点的连线是可视线，那么称该组监视器覆盖多边形 P。图 2-9 中 4 只监视器覆盖 12 个顶点的多边形。可以证明，覆盖 12 个顶点的任意多边形需要监视器的最大数恰好是 4，也就是说，4 只监视器总可以覆盖 12 个顶点的任意多边形。

设 $f(P_n)$ 是覆盖有 n 个顶点的任意多边形 P_n 所需监视器的最小数

$$f(P_n) = \min_S |\{S|S \text{ 覆盖 } P\}|$$

其中，S 是点集；$|S|$ 表示 S 中元素的个数。另设 $F(n)$ 表示所有 n 个顶点多边形 P_n 上函数 $f(P_n)$ 的最大值

$$F(n) = \max_{P_n} f(P_n)$$

问题是要确定函数 $F(n)$，$F(n)$ 对于所有的 n 是有限的。容易得到 $F(n)$ 的平凡上、下界为

$$1 \leqslant F(n) \leqslant n$$

这表明覆盖多边形 P 至少需要 1 只监视器，最多不超过 n 只监视器。

$n=3$ 时，P 是三角形，只需要 1 只监视器，即 $F(3)=1$。

$n=4$ 时，P 是四边形，四边形分两种情况：凸四边形和有一个凹点的四边形。对于凸四边形，只需要 1 只监视器，而对于有一个凹点的四边形也只需要 1 只监视器，该监视器安装在凹点附近适当的位置，因此 $F(4)=1$。

对于五边形，有三种情况：凸的、1 个凹点和两个凹点。无论是哪一种情况，均只需 1 只监视器，所以 $F(5)=1$。

六边形就复杂些，它不仅会有两个凹点，而且凹点的互相位置及与凹点关联的两条边的方向也变得复杂了。可能有 $F(6)=2$，如图 2-10 所示。

将图 2-10 中 $n=6$ 的一种情况推广，得到图 2-11 所示的多边形。它有多个叉子，每个

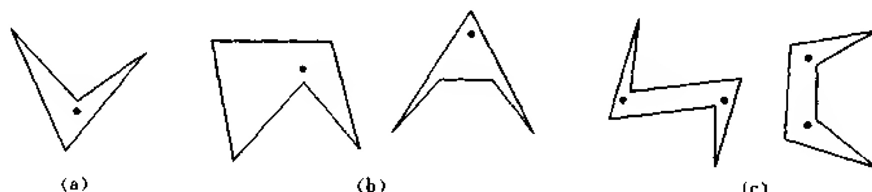


图 2-10 $n=4,5,6$ 个顶点的多边形及所需要的监视器

叉子需要 1 只监视器，而每个叉子由 3 条边组成。因此，设多边形有 n 条边，组成 $\frac{n}{3}$ 个叉子，那么需要 $\frac{n}{3}$ 只监视器才能覆盖该多边形，即推得 $F(n)\geqslant\frac{n}{3}$。这是利用一般的例子来建立 $F(n)$ 的下界。由上述实例推测 $F(n)=\frac{n}{3}$。

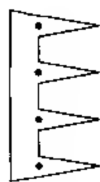


图 2-11　推导 $F(n)\geqslant\frac{n}{3}$ 的例子

定义 2-8　设多边形 P 的顶点序列是 $p_1, p_2, \cdots, p_n$，在可视线定义中，令 $x=p_i, y=p_j, j\neq i, i+1$，则线段 $\overline{p_ip_j}$ 称为 P 的对角线。

定义 2-9　如果 $\overline{ab}$、$\overline{cd}$ 是多边形 P 的两条对角线，并且 $\overline{ab}\cap\overline{cd}$ 为空或在端点相交，则称 $\overline{ab}$ 与 $\overline{cd}$ 是两条不交叉的对角线。

利用不交叉的对角线划分多边形成若干个三角形，然后对各三角形的三个顶点着不同的三种颜色，最后选着同一颜色的顶点放置监视器。这是证明下面定理的主要思想。

定理 2-6　$F(n)=\lfloor n/3\rfloor$。

证明　对任意多边形 P 都可以用不交叉的对角线划分 P 成若干个三角形，称为 P 的三角剖分。任意多边形 P 的三角剖分的存在性将在 2.3 节中证明，这里假设已经将 P 三角剖分(三角剖分的算法也将在 2.3 节中介绍)。

显然，三角剖分之后所得到的图 G，其弧是多边形的边和三角剖分的对角线，而结点是多边形 P 的顶点。图 G 是 3 可着色的，即对 G 的结点赋 3 种颜色，使得任一条弧的两个端点均着不同颜色。这样，三角剖分 P 的所有三角形的三个顶点可以着不同颜色。这是由于此项着色工作是在多边形顶点上进行，并且三角剖分 P 的对角线不交叉的缘故。

设红、蓝和黄是对图 G 的结点进行着色的三种颜色，G 中每个三角形的 3 个顶点必有 3 种不同颜色。因此，每个三角形必有 1 个顶点着红色。假设在每个红色顶点处放置 1 只监视器，那么每个三角形在一个顶点上有 1 只监视器。显然三角形被在它的一个顶点上的监视器覆盖，因此 G 中每个三角形都被覆盖了，从而多边形 P 也完全被覆盖。

由于多边形 P 有 n 个顶点，而且对顶点仅使用 3 种颜色着色，故由鸽巢原理，可以断定，1 种颜色仅使用 $\lfloor n/3\rfloor$ 次。因此，将使用不多于 $F(n)=\lfloor n/3\rfloor$ 只监视器覆盖多边形。

证毕。

定理 2-6 回答了问题 2-1，其证明过程也是求解该问题的一种算法。上述讨论不一定导致最少监视器数目，如图 2-12 所示，$n=14$，颜色 3 只重复使用三次。这是由于上述讨论中，应用鸽巢原理时，隐含了一个假设：n 只鸽子均分到三个鸽巢内。

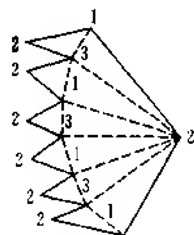


图 2-12　$n=14$ 的多边形，只需 3 只监视器

问题 2-2　确定任意多边形的核

定义 2-10　给定任意简单多边形 P，若 P 内存在点 z，使得对于 P 内的所有点 p，线段 $\overline{zp}$ 完全位于 P 内。具有上述性质的点 z 的轨迹称为多边形 P 的核。

如果 P 的顶点序列按逆时针方向排列，即 P 的内部在其边的左侧，那么 P 的核是其边的左半平面的交。

构造多边形的核不仅是解决某些几何问题(比如包含问题)所需要的预处理步骤，而

且是某些工程设计、计算机图形学等领域中不可回避的问题。Lee 和 Preparata 提出了一种算法，它依次扫描多边形 P 的顶点，且构造凸多边形序列 $K_1, K_2, \cdots, K_{n-1}$，其中 $K_{n-1} = K(P)$ 即所要求的多边形的核。下面是周培德提出的一种算法，该算法只扫描并处理 P 的凹点（所得多边形序列 $P_2, P_3, \cdots, P_l$ 不一定是凸的，而 P_{l+1} 是凸的，即所求的核），对于 P 的凸点则不作什么处理，因而节省了时间。

$Z_{2\text{-}1}$ 算法（确定多边形核的算法）

输入　多边形 P 的顶点序列 $p_1, p_2, \cdots, p_n$（逆时针排列）。

输出　多边形 P 的核 $K(P)$ 的顶点序列。

步 1　确定多边形 P 的凸凹顶点。

步 2　**if**　$p_i(i=\overline{1,n})$ 是凸的　**then**　$K(P)=P$

else　设 $p_1, \cdots, p_s, \cdots, p_t, \cdots, p_m$ 是凹的，重新编码凹点为 $P' = \{q_1, q_2, \cdots, q_l\}$，$q_1 = p_1, q_2 = p_s, \cdots, q_l = p_m$。

步 3　$i \leftarrow 1, P_1 = P$

步 4　求与 $q_i(=p_{i+h})$ 关联的两条边（$l_1 = \overline{p_{i+h-1}p_{i+h}}$ 和 $l_2 = \overline{p_{i+h}p_{i+h+1}}$）和 P_i 的边的交点。

步 5　**if**　l_1 与 P_i 的边的交点数是 1（设为 I_i）$\wedge l_2$ 与 P_i 的边的交点数是 1（设为 $I_{i'}$）$\wedge I_i(I_{i'})$ 在 $l_2(l_1)$ 的左侧

then　q_i, I_i, I_i 与 $I_{i'}$ 之间 P_i 的顶点，$I_{i'}$ 构成 P_{i+1}。在 P' 中删去 q_i。$i \leftarrow i+1$，**goto**　步 4，直至 $i>l$，$K(P)=P_i$，终止。

步 6　**if**　l_1 与 $P_i(i>1)$ 的边的交点数是 0（或 2，设 $I_i, I_{i'}$ 是交点 $\wedge I_i, I_{i'}$ 在 $l_2 = \overrightarrow{P_{i+h}P_{i+h+1}}$ 的左侧）$\wedge l_2$ 与 $P_i(i>1)$ 的边的交点数是 2（设 $I'_i, I'_{i'}$ 是交点 $\wedge I'_i, I'_{i'}$ 在 $\overrightarrow{P_{i+h-1}P_{i+h}}$ 的左侧）（或 0）$\wedge |\overline{P_{i+h}I_i}| < |\overline{P_{i+h}I_{i'}}|$（$|\overline{P_{i+h}I'_i}| < |\overline{P_{i+h}I'_{i'}}|$）

then　从 P_i 中删去 $\overrightarrow{I_iI_{i'}}$ 右侧（$\overrightarrow{I'_iI'_{i'}}$ 左侧）部分，构成 P_{i+1}。在 P' 中删去 q_i。$i \leftarrow i+1$，**goto**　步 4，直至 $i>l$，$K(P)=P_i$，终止。

步 7　**if**　l_1 与 $P_i(i>1)$ 的边的交点数是 1（设 I_i 是交点 $\wedge I_i$ 在 $\overrightarrow{P_{i+h}P_{i+h+1}}$ 的左侧）或 $0 \wedge l_2$ 与 $P_i(i>1)$ 的边的交点数是 0 或 1（设 I_i 是交点 $\wedge I_i$ 在 $\overrightarrow{P_{i+h-1}P_{i+h}}$ 的左侧）

then　从 P_i 中删去 $\overrightarrow{P_{i+h}I_i}$ 右（左）侧部分，构成 P_{i+1}。在 P' 中删去 q_i。$i \leftarrow i+1$，**goto**　步 4，直至 $i>1$。$K(P)=P_i$，终止。

步 8　**if**　l_1 与 P_i 的边的交点数是 1 或 3（分别设为 I_1；$I_1, I_2, I_3 \wedge$ 交点在 $\overrightarrow{P_{i+h}P_{i+h+1}}$ 的左侧）$\wedge l_2$ 与 P_i 的边的交点数是 3 或 1（分别设为 I'_1, I'_2, I'_3；$I'_1 \wedge$ 交点在 $\overrightarrow{P_{i+h-1}P_{i+h}}$ 的左侧）$\wedge |\overline{P_{i+h}I_1}| < |\overline{P_{i+h}I_2}| < |\overline{P_{i+h}I_3}|$（或 $|\overline{P_{i+h}I'_1}| < |\overline{P_{i+h}I'_2}| < |\overline{P_{i+h}I'_3}|$）

then　$I_1(I'_1)$ 与 $I_2(I'_2)$ 之间存在一个凹点，设为 $q_a(a>i)$。$q_i, I_1, q_a, I_2, I_3, I_3$ 与 I'_1 之间的 P_i 顶点，I'_1（或 $q_i, I'_1, q_a, I'_2, I'_3, I'_3$ 与 I_1 之间的 P_i 顶点，

I_1)构成 P_{i+1}。求与 q_a 关联的两条边($l_1'=\overline{p_{a+K'-1}p_{a+K'}}$ 和 $l_2'=\overline{p_{a+K'}p_{a+K'+1}}$)和 P_{i+1}的边的交点。

if l_1' 与 P_{i+1}交于 I_4,l_2' 与 P_{i+1}交于 $I_5 \wedge I_4$ 在$\overrightarrow{p_{a+K'}p_{a+K'+1}}$左侧 $\wedge I_5$ 在$\overrightarrow{p_{a+K'-1}p_{a+K'}}$左侧

then q_a,I_4,I_4 与 I_5 之间的 P_{i+1}顶点,I_5 构成 P_{i+2}。在 P'中删去 q_a 与 q_i。$i<j<a$ 时,$q_j \leftarrow q_{j+1}$。$i \leftarrow i+2$,**goto** 步 4,直至 $i>1$。$K(P)=P_i$,终止。

else 重复步 6 或步 7

步 9 if 与 q_i 关联的两条边 l_1,l_2 和 $P_i(i>1)$的边的交点(交点互处对方边的左侧)数均为 $0 \wedge P_i$ 的全部顶点在$\overrightarrow{P_{i+k-1}P_{i+k}}$的左侧和$\overrightarrow{P_{i+k}P_{i+k+1}}$的左侧

then $P_{i+1} \leftarrow P_i$,在 P' 中删去 q_i,$i \leftarrow i+1$,**goto** 步 4,直至 $i>1$。$K(P)=P_i$,终止。

else if 与 q_i 关联的两条边和 P_i 的边的交点(交点互处对方边的左侧)数均为 $0 \wedge P_i$ 的全部顶点在$\overrightarrow{p_{i+k-1}p_{i+k}}$的右(左)侧 $\wedge P_i$ 的全部顶点在$\overrightarrow{p_{i+k}p_{i+k+1}}$的左(右)侧

then $K(P)=\varnothing$,终止。

对 $Z_{2\text{-}1}$算法稍作修改,便可用来求解问题 2-1,这里不赘述了,作为一道习题留给读者。

根据多边形核的定义,凸多边形的核显然是其自身。当多边形 P 有一个凹点时,请见图 2-13,$q_1(=p_4)$是凹点,$\overrightarrow{p_3p_4}$的延长线与 P 的边$\overline{p_8p_9}$的交点为 I_i,I_i 在$\overrightarrow{p_4p_5}$的左侧。$\overline{p_4p_5}$的延长线与$\overline{p_9p_{10}}$的交点 I_r 位于$\overrightarrow{p_3p_4}$的左侧。$p_4(=q_1)$,I_i,p_9 和 I_r 围成的多边形 P_2 即所求的核。因为 P_2 中的点满足核定义中对 z 点的要求,即域 P_2 是点 z 的轨迹。这也表明,P_2 是从 $P_1(=P)$中割去不符合 q_1 要求的区域得到的。

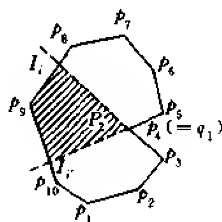


图 2-13 具有 1 个凹点多边形 P 的核 P_2

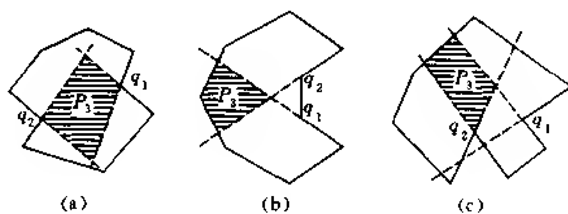


图 2-14 具有 2 个凹点多边形 P 的核 P_3

设 P 中有 l 个凹点。当 $l=2$ 时,对 q_2 按 $Z_{2\text{-}1}$算法所示步骤可以求出核 P_3,图 2-14 中(a)、(b)、(c)表示三种情况。P_3 是从 P_2 中割去不符合 q_2 要求的区域得到的,因此,$P_2 \supseteq P_3$。

如果对 $l-1$ 个凹点已求得 P_l,则对 q_l 再重复一次步 4 和步 5(或步 6 或步 7 或步 8 或步 9),即从 P_l 中再割去不符合 q_l 要求的区域,便得到多边形 P 的核 P_{l+1}。显然有 $P_1 \supseteq P_2 \supseteq \cdots \supseteq P_l \supseteq P_{l+1}$。总之,该算法是对凹点逐个求出 P_i 的,位于 P_{l+1}中的点满足核定义中对 z 点的要求,即 P_{l+1}中的点与 P 中所有点 p 的连线完全位于 P 内。因此,$Z_{2\text{-}1}$算法正确

地求得了 P 的核。

步 1 只要用线性次乘法便可确定多边形 P 各顶点的凸凹性，并记住其凸凹性。步 2 不需另耗时间。步 3 用常数时间。步 4 判断 P_i 的边的两个端点是否在 l_1、l_2 的两侧，如果在两侧，则求交点，这需要线性次乘法。步 5 进行逆时针查找 I_r 与 $I_{r'}$ 之间 P_i 的顶点，所需时间可以不计。步 5 至步 4 至多循环 l 次，所需乘法次数为 $O(ln)$。步 6 至步 4，步 7 至步 4，步 8 至步 4，步 9 至步 4 等的循环均不可能超过 l 次，所需乘法次数不超过 $O(ln)$，比较次数可以不计。因此，$Z_{2\cdot1}$ 算法的总耗费为 $O(ln)$ 次乘法，其中 l 是多边形 P 的凹点的数目。

2.3 多边形的三角剖分

本节首先证明任意简单多边形都存在三角剖分，然后介绍三角剖分的某些基本性质和构造三角剖分的算法。

定义 2-11 设多边形 P 有 n 个顶点 $p_1, p_2, \cdots, p_n$，$\overline{p_ip_j}$ 是 P 的对角线，并且不与 P 的顶点及边相交，它划分 P 为两部分。对 P 逐步增加不交叉的对角线，直至 P 的内部全部被划分成三角形，这样的划分称为多边形的三角剖分。

证明多边形都存在一个三角剖分的关键是证明对角线的存在，为此需要一个事实：每个多边形必定至少有一个凸顶点。

定理 2-7 每个多边形至少有一个凸顶点。

证明 如果多边形的顶点 $p_1, p_2, \cdots, p_n$ 按逆时针方向排列，并且 y 坐标值最小的顶点设为 p_i(唯一)，那么 p_{i+1} 必定在 $\overrightarrow{p_{i-1}p_i}$ 的左侧。因此点 P_i 为凸点。考虑 y 坐标值最大的顶点及 x 坐标值最大、最小的顶点，由于同样的原因，它们也必定为凸点。 证毕。

定理 2-8 顶点数 $n \geqslant 4$ 的多边形至少有一条对角线。

证明 由定理 2-7，多边形至少有一个凸顶点，设 p_i 是一个凸顶点，p_{i-1} 和 p_{i+1} 是与 p_i 相邻的顶点。如果 $\overline{p_{i-1}p_{i+1}}$ 是一条对角线，则定理成立。否则，假定 $\overline{p_{i-1}p_{i+1}}$ 不是一条对角线，则或者 $\overline{p_{i-1}p_{i+1}}$ 在 P 的外部，或者它与 ∂P 相交。在这两种情况的任一情况下，既然 $n>3$，三角形 $p_{i-1}p_ip_{i+1}$ 至少包含 P 的一个顶点(除了 p_{i-1}, p_i, p_{i+1} 之外)，设 p 是距 p_i 最近的，并在三角形 $p_{i-1}p_ip_{i+1}$ 内部的 P 的顶点，该顶点是从 p_i 向 $\overline{p_{i-1}p_{i+1}}$ 移动并平行于 $\overline{p_{i-1}p_{i+1}}$ 的线 L 所碰到的三角形 $p_{i-1}p_ip_{i+1}$ 中的第一个顶点。如图 2-15 所示。

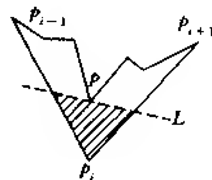


图 2-15 定理 2-8 证明的示意图

显然，由与包括 p_i(图 2-15 中的阴影域)的 L 所限界的半平面相交的三角形 $p_{i-1}p_ip_{i+1}$ 的内部不再包含 ∂P 的顶点。因此，除了在 p_i 和 p，$\overline{p_ip}$ 不可能与 ∂P 相交，即 $\overline{p_ip}$ 是一条不与 ∂P 相交的对角线。 证毕。

定理 2-9 设多边形 P 具有 n 个顶点，对 P 通过添加零条或者多条对角线可以剖分成三角形。

证明 对多边形顶点数 n 施归纳法。如果 $n=3$，多边形 P 是一个三角形，定理显然

成立。设$n\geq4$，由定理 2-8，假定$d=\overline{p_ip_j}$是P的一条对角线，另由对角线的定义，d仅在其端点与∂P相交，并把P划分成两个多边形P'与P''，这两个多边形以d作为一条公共边，而且每个多边形的顶点数少于n，如图 2-16 所示，这是由于该剖分过程中没有增加新的顶点。显然，在P'与P''中除了p_i与p_j之外至少有一个顶点。对P'与P''应用归纳假设便完成了定理的证明。

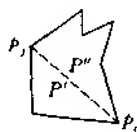


图 2-16　对角线划分 P 为 P' 与 P''

下面介绍三角剖分的基本性质。

定理 2-10　n个顶点的多边形P的任一三角剖分使用$n-3$条对角线，并且由$n-2$个三角形组成。

证明　用归纳法证明。$n=3$时，用 0 条对角线并有 1 个三角形，定理成立。

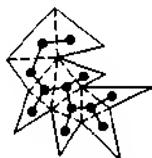
$n\geq4$，设对角线$d=\overline{p_ip_j}$划分P成两个多边形P'和P''，并且它们分别有n_1和n_2个顶点，$n_1+n_2=n+2$，这是由于p_i与p_j被重复计数 1 次。对P'和P''应用归纳假设，得到$(n_1-3)+(n_2-3)+1=n-3$条对角线，其中项"$+1$"是计数对角线d，并且有$(n_1-2)+(n_2-2)=n-2$个三角形。　证毕。

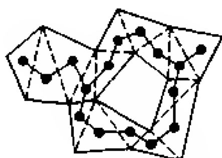
推论 2-1　n个顶点多边形的内角之和是$(n-2)\pi$。

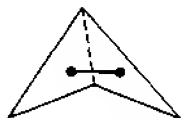
证明　由定理 2-10，三角剖分P得到$n-2$个三角形，而每个三角形内角之和为π，故推论成立。

下面引入三角剖分对偶的概念。

定义 2-12　多边形P三角剖分的对偶T是一个图，三角剖分中的每个三角形用一结点表示，三角剖分中有相邻边的两个三角形，则所表示的结点p_i和p_j连以线$\overline{p_ip_j}$。如图 2-17所示。


图 2-17　三角剖分的对偶


图 2-18　T有一条回路


图 2-19　T有两个耳

定理 2-11　无孔简单多边形三角剖分的对偶T是一棵树，并且该树的每个结点的度数至多为 3。

证明　由于三角剖分中每个三角形至多有三条边共享，所以图T中每个结点的度数至多为 3。

假定T不是树，则它必有一条回路C，组成该回路的结点所对应的三角形必形成多边形内的一个孔，如图 2-18 所示，这与已知条件矛盾，因此T是一棵树。　证毕。

度数为 1 的结点是T的叶；度数为 2 的结点位于树的路径上；度数为 3 的结点是分支点。当根选在度数为 1 或者 2 的任意结点处时，T是一棵二叉树。三角剖分的对偶与二叉树之间的这种对应关系是十分有用的。

定义 2-13　设p_{i-1}、p_i、p_{i+1}是多边形P的三个连续顶点，并且$\overline{p_{i-1}p_{i+1}}$是$P$的一条对角线，那么$p_{i-1}$、$p_i$、$p_{i+1}$构成$P$的一个耳，$p_i$称为耳尖。

定理 2-12 顶点数 $n \geqslant 4$ 的任意多边形至少有两个不重叠的耳。

证明 三角剖分对偶 T 中的一个叶结点对应于一个耳。具有两个结点的树，如图 2-19所示，有两个不重叠的耳。由定理 2-10，具有多个结点的树，有 $(n-2)>2$ 个结点，必至少有两个叶。 证毕。

依据定理 2-12，并利用不断删去一个耳的方法，可以对三角剖分图进行 3 着色。

定理 2-13 多边形 P 的三角剖分图可以用 3 种颜色着色。

证明 对顶点数 n 施归纳法证明。$n=3$，三角形可以 3 着色。

$n \geqslant 4$，由定理 2-12，P 有一个耳，设为三角形 $p_{i-1}p_ip_{i+1}$，其中 p_i 是耳尖。切割耳 $p_{i-1}p_ip_{i+1}$，构成一个新的多边形 P'：用 $\partial P'$ 中的 $p_{i-1}p_{i+1}$ 代替 ∂P 中的序列 $p_{i-1}p_ip_{i+1}$。P' 有 $n-1$ 个顶点，由归纳假设，P' 可以 3 着色。对三角形 $p_{i-1}p_ip_{i+1}$ 中的 p_i，用不同于对 p_{i-1}、p_{i+1} 着的颜色来着色 p_i，这就是 P 的 3 着色。 证毕。

下面介绍任意简单多边形三角剖分的一种算法，该算法所得到的三角剖分，其对角线的长度之和可以达到最小或次最小，即最小权或次最小权三角剖分。基本想法是先将任意简单多边形划分成若干个凸多边形，然后对每个凸多边形进行三角剖分。利用不断切去一个耳的方法可以将凸多边形三角剖分，但不一定能得到最小权三角剖分。

$Z_{2\text{-}2}$算法（凸多边形三角剖分的算法 CPTA）

输入 凸多边形 P_1 的顶点序列 $p_1, p_2, \cdots, p_n$。

输出 P_1 的三角剖分序列 $p_1p_2p_i, p_2p_ip_j, \cdots, p_ip_{n-1}p_n$。

步 1 计算 P_1 的直径，设直径的两个端点为 p_i 与 p_j。

步 2 比较 $\overline{p_{i-2}p_i}, \overline{p_{i-1}p_{i+1}}, \overline{p_ip_{i+2}}$ 的长度，取较短者作为对角线，删去相应的顶点（耳尖），并输出相应的三角形（耳）。对 p_j 同样处理。

步 3 $P_1' \leftarrow P_1-(p_{i-1} \cup p_i \cup p_{i+1}) \cap (p_{j-1} \cup p_j \cup p_{j+1})$

步 4 对 P_1' 重复步 1 至步 4，直至 P_1 被分割完毕。

利用算法 $Z_{3\text{-}6}$ 计算凸多边形的直径，耗费 $O(n)$。步 2 与步 3 只需要常数时间。步 4 至步 1 至多循环 $\left\lfloor \frac{n-2}{2} \right\rfloor$ 次，因此算法 $Z_{2\text{-}2}$ 的耗费为 $O(n^2)$。

如果不考虑最小权三角剖分，而只是要求三角剖分凸多边形，那么不需要步 1 与步 2 中的长度比较。这时只需不断切割凸多边形的耳，复杂性为 $O(n)$。

$Z_{2\text{-}3}$算法（任意多边形三角剖分的算法 APTA）

步 1 分割多边形 P 成凸多边形序列 $P_1, P_2, \cdots, P_k$。

步 2 对 $P_i(i=\overline{1,k})$ 执行 $Z_{2\text{-}2}$ 算法

利用算法 $Z_{2\text{-}4}$ 完成步 1，耗费 $O(n)$，因此 $Z_{2\text{-}3}$ 算法的耗费为 $O(n^2)$。如果步 2 改为不断切割凸多边形的耳，那么 $Z_{2\text{-}3}$ 算法的复杂性为 $O(n)$。

2.4 多边形的凸划分

本节介绍单调多边形的三角剖分，以及简单多边形划分成梯形和凸多边形（简称凸划分）的算法。

定义 2-14 设链 C 是由多边形边 $e_i, e_{i+1}, \cdots, e_j$ 组成的多边形链，L 是平面上一条直线，如果与 L 垂直的直线 L' 与 C 至多交于 1 点，即 $L' \cap C$ 或者为空，或者为单个点，则称多边形链 C 关于 L 是单调的。

我们取 y 轴作为直线 L。给定多边形 P 的顶点序列 $p_1, p_2, \cdots, p_n$，求其 y 坐标最大、最小的顶点，设为 p_i, p_j。p_i, p_j 分割多边形 P 的顶点序列为两个子顶点序列，记为 P_1 与 P_2。P_1 与 P_2 关于 y 轴的单调性取决于组成它们的点序列的 y 坐标是否为单调的。图 2-20(a)所示的多边形链 C_1 与 C_2 关于 y 轴是单调的，而图 2-20(b)所示的多边形链 C_1' 与 C_2' 关于 y 轴不是单调的。

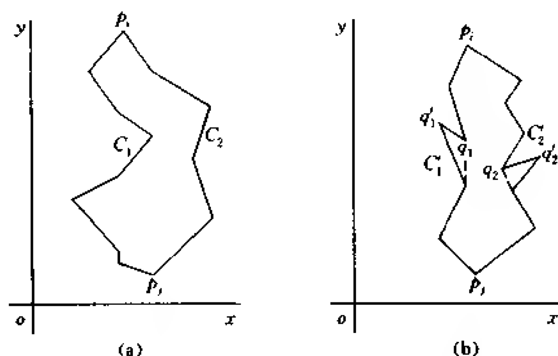


图 2-20 多边形链关于 y 轴的单调性

定义 2-15 设 p_{i-1}, p_i 与 p_{i+1} 是多边形 P 的三个相邻的顶点，并且 p_{i-1}, p_{i+1} 的 y 坐标小于(大于)或等于 p_i 的 y 坐标，则称 p_i 为 P 的歧点。图 2-20(b)中 q_1, q_1', q_2 与 q_2' 都是歧点。具有歧点的链不是单调链，图 2-20(b)中链 C_1' 与 C_2' 不是单调链。

如果 p_i 是 P 的歧点，并且 p_i 是 P 的凸点(耳尖)，那么利用切割耳的方法可以删去歧点 p_i。如图 2-20(b)中 q_1' 与 q_2' 可以被删去，使链 C_1' 与 C_2' 成为单调链。

定义 2-16 如果组成链 C 的顶点均为多边形 P 的凹点，则链 C 称为 P 的凹链。图 2-21中链 C_2 由 $\overline{p_1p_2}$、$\overline{p_2p_3}$、$\overline{p_3p_4}$、$\overline{p_4p_5}$ 组成，并且顶点 p_2、p_3 与 p_4 均为凹点，因此链 C_2 是一条凹链。顶点 p_6 在与 C_2 相对的链 C_1 上，并且 p_6 的 y 坐标小于 p_4、p_3 与 p_2 的 y 坐标，此时连接 p_6 与 p_4，p_6 与 p_3，p_6 与 p_2，便将多边形 P 三角剖分。

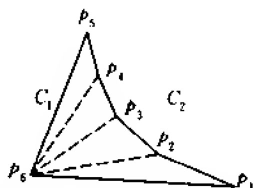


图 2-21 凹链及其三角剖分

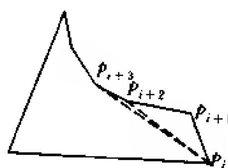


图 2-22 p_i 与凹链共链

如果 p_i 与凹链共链，如图 2-22 所示，p_{i+1} 是凸点，此时连接 p_i 与 p_{i+2}，p_i 与 p_{i+3}，直至 p_{i+j} 位于 $\overrightarrow{p_ip_{i+j-1}}$ 的右侧时停止连接。

如果 p_i 与凹链共链，并且 p_{i+1}是凹点，则将 p_i 加入凹链。

下面介绍简单多边形三角剖分的一种算法，其思想是先采用切割耳的方法删去歧点（同时又是凸点），使多边形的两条链 C_1 与 C_2 成为单调链，然后三角剖分单调多边形。

简单多边形三角剖分的算法

输入　多边形 P 的顶点序列 $p_1,p_2,\cdots,p_n$ 按逆时针方向排列。

输出　三角剖分 P 的三角形序列。

步 1　**if**　p_i 是 P 的歧点 ∧ p_i 是凸点。

　　then　连接 p_{i-1}与 p_{i+1}，删去 p_i，输出三角形 $p_{i-1}p_ip_{i+1}$。

步 2　按 y 坐标分类 P 顶点成递增序列：$p_b,\cdots,p_c,p_{c-1},p_a$。$p_b$、$p_a$ 分割 ∂P 为左、右链 C_1,C_2。

步 3　预置凹（凸）链是两个顶部顶点。

步 4　设 p_c 是第 3 个最高的顶点。

步 5　**while**　$p_c \neq p_b$　**do**

　　if　p_c 是在与凹（凸）链相对的链上

　　　then　由 p_c 到凹（凸）链顶点 p_{c-1}画对角线，并删去凹（凸）链顶部顶点。

　　　　if　凹（凸）链是空　**then**　下移顶点

　　else　p_c 邻接于凹（凸）链的底

　　　if　p_{c-1}是凸点

　　　　then　由 p_c 至凹（凸）链的底 p_{c-2}画对角线，并删去凹（凸）链的底。

　　　　　if　凹（凸）链是空　**then**　下移顶点

　　　else　p_{c-1}是凹点，将 p_c 添加到凹（凸）链的底，下移顶点。

该算法步 1 需要线性时间，步 2 耗费 $O(n\log n)$，步 3 与步 4 只要常数时间，步 5 需要线性时间，因此，算法的时间复杂性为 $O(n\log n)$。

图 2-23 所示的多边形是一个有 25 个顶点的单调多边形，由于图中没有歧点，所以跳

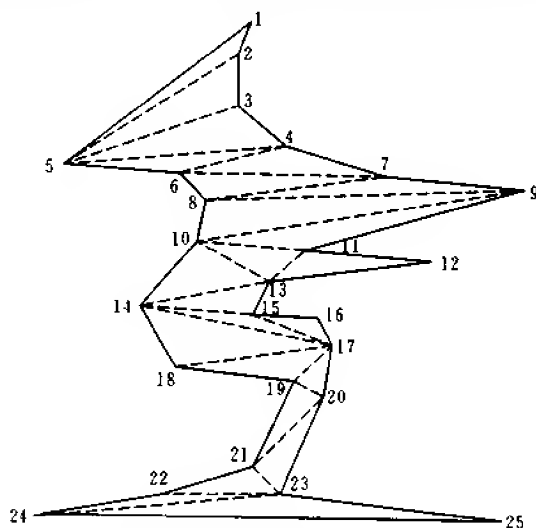


图 2-23　单调多边形的三角剖分

过步 1。步 2 按 y 坐标分类顶点，得到点序 1，2，…，25 及左链 1，5，6，8，10，14，18，19，21，22，24，25；右链 1，2，3，4，7，9，11，12，13，15，16，17，20，23，25。点 3 邻接于凹链 1，2 的底，并且点 2 是凹点，所以将点 3 添加到凹链，得到凹链 1，2，3。同理将点 4 加入凹链，得到凹链 1，2，3，4。点 5 在与凹链 1，2，3，4 相对的链上，那么点 5 分别与点 2，3，4 连接。下移顶点到点 6，连接 6 与 4。当下移到点 12 时，凹链是 10，11，由于 11 是凹点，故将 12 加入到凹链，得到凹链 10，11，12。当下移到点 13 时，由于 12 是凸点，则连接 13 与 11，13 与 10。继续下去，直至分割完毕。

现在介绍划分多边形成梯形的算法，划分多边形成梯形是三角剖分多边形的一种中间过程。划分多边形成梯形的算法思想如下：通过多边形的每个顶点画水平线，这些水平线与多边形边相交，位于多边形内部的线段记为 s，即 $s \subset P$，并且 $s \cap \partial P = \{p_i, q_j\}$，其中 p_i 是 P 的顶点，q_j 是 s 与多边形边的交点，s 将多边形划分成梯形。可能出现 $s \cap \partial P = \{p_i\}$ 的退化情况或者 $s \cap \partial P = \{p_i, q_j, q_{j+1}, \cdots, q_k\}$。假设多边形 P 没有两个顶点位于一条水平线上。图 2-24 是划分多边形成梯形的一个例子，其中点线起着删去某些点的作用，这些点既是 P 的歧点又是 P 的凸点，从而使 P 成为单调多边形；虚线是梯形划分线。

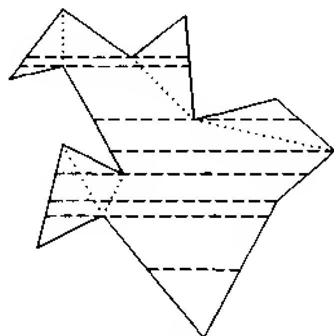


图 2-24　多边形划分成梯形

划分多边形成梯形的算法中使用了"平面扫描"技术，主要思想是用一条水平线"扫描"平面，并且在水平线保持某种类型的数据结构。在离散事件处，即在出现新的顶点处停止扫描，并修改数据结构。划分多边形成梯形时，在顶点 p_i 处水平线暂停平移（通过按 y 坐标对顶点进行排序来实现），寻找 p_i 左边的一条边和 p_i 右边的一条边。为了有效地完成这个工作，保持水平线（图 2-25 中所示扫描线 L）穿过多边形边的顺序表，该表是 $A=(e_{19}, e_{18}, e_{17}, e_7, e_9, e_{11})$。通过检查各边端点 y 坐标是否跨越 L，可以确定表 A 由哪些边组成，然后对表 A 中各边端点按 x 坐标排序，又可以确定点 p_i 位于 e_{17} 与 e_7 之间。如果把表 A 存储在 2-3 树中，那么搜索 p_i 的位置仅需要 $O(\log n)$ 时间。这也是处理一个事件所需要的时间，因此整个平面扫描的总耗费是 $O(n\log n)$。

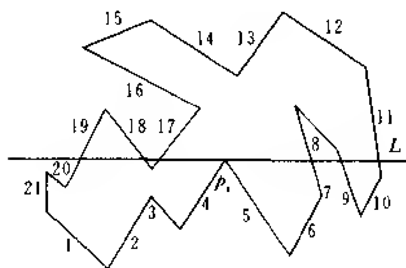


图 2-25　平面扫描、标记检索边

图 2-25 中，当 L 在多边形上方时，L 穿过多边形边的表是空，而当 L 经过每个事件顶点时表示成下述表列：(e_{13}, e_{12})，(e_{14}, e_{13}, e_{12})，(e_{16}, e_{14}, e_{13})，(e_{16}, e_{11})，(e_{16}, e_{11})，(e_7, e_8, e_{11})，$(e_{17}, e_7, e_8, e_{11})$，$(e_{19}, e_{18}, e_{17}, e_7, e_{11})$，$(e_{19}, e_{18}, e_{17}, e_7, e_9, e_{11})$ 等。

首先划分多边形成梯形，然后再划分成三角形，其算法如下。

多边形三角剖分算法

步 1　按 y 坐标对多边形的顶点进行排序。

步 2　用平面扫描方法划分多边形成梯形。

步 3　用删去既是歧点又是凸点的点的方法构造单调多边形。

步 4 在梯形或相邻梯形中用连接多边形顶点的方法划分梯形成三角形。

步 1 与步 2 分别耗费时间 $O(n\log n)$，步 3 与步 4 分别耗费 $O(n)$时间，所以算法的总耗费为 $O(n\log n)$。

1991 年，Chazelle 提出一种三角剖分多边形的算法，其复杂性为 $O(n)$。算法中引入可视映射的概念，它把梯形改变为朝多边形链每个顶点的两边画水平线，然后模拟合并分类。n 个顶点的多边形划分成具有$\frac{n}{2}$个顶点的链，然后再把这些链划分成$\frac{n}{4}$个顶点的链等等，通过合并子链的映射找到链的可视映射。这种方法的复杂性为 $O(n\log n)$。后来，Chazelle 改进上述过程成两个阶段，第一阶段求近似可视映射，耗费为 $O(n)$；第二阶段把近似映射改进成完全的可视映射，耗费也是 $O(n)$。然后由可视映射所定义的梯形产生三角剖分。

划分多边形成三角形是划分多边形成凸多边形的一种特殊情况。划分多边形成凸多边形有两个目标：(1) 划分多边形成尽可能少的凸多边形；(2) 尽可能快地完成划分工作，即设计时间复杂性低的算法。

用来划分多边形的线段有两种：多边形的对角线；端点在 ∂P 上并位于 P 内部的线段。线段划分较对角线划分更复杂些。凸多边形的数目与凹点的数目有下列关系。

定理 2-14 设 M 是划分多边形成凸多边形的最少数目，对于有 r 个凹点的多边形，则有 $\left\lceil \frac{r}{2} \right\rceil +1 \leqslant M \leqslant r+1$。

证明 对多边形的每个凹点画一条角平分线，因而得到凸划分，其凸多边形的数目为 $r+1$，如图 2-26 所示。另外，一条对角线至多可以分解两个凹点，这便产生 $\left\lceil \frac{r}{2} \right\rceil +1$ 个凸多边形，如图 2-27 所示。

证毕。

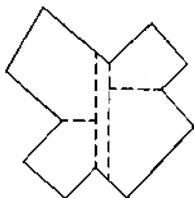


图 2-26 定理 2-14 证明示意图，$r=4$，$M=5$

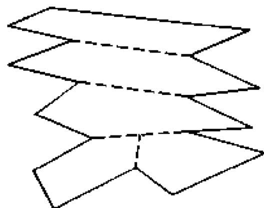


图 2-27 定理 2-14 证明示意图，$r=7$，$M=5$

定义 2-17 在由对角线划分多边形成凸多边形的划分中，如果在顶点 p_i 处删去对角线 d 之后产生非凸多边形，则称对角线 d 关于顶点 p_i 是基本的。对顶点 p_i 不是基本的对角线称为非基本的。

Hertel 和 Mehlhorn 于 1983 年提出了一种划分多边形成凸多边形的算法，其基本思想是，从 P 的三角剖分开始，重复删去非基本对角线。该算法显然在线性时间内可以完成。可以证明 H-M 算法产生凸多边形的数目小于 $4M$。

寻找最优凸多边形数目的凸剖分比寻找次最优费时间得多。1983 年，Greene 设计出最优凸剖分的一种算法，时间复杂性为 $O(n^4)$。1985 年，Keil 改进到 $O(n^3\log n)$。

如果用线段而非对角线进行剖分，那么问题就困难了。1980 年 Chazelle 设计出复杂性为 $O(n^3)$的算法解决了这个问题。

本节最后介绍周培德于 1997 年提出的一种剖分多边形成凸多边形的算法，其时间复杂性是 $O(n)$。

定义 2-18 如果 p_i 是多边形 P 的凹点，$\overline{p_{i-1}p_i}$、$\overline{p_ip_{i+1}}$是与 p_i 关联的两条边，$\overline{p_{i-1}p_i}$($\overline{p_{i+1}p_i}$)的延长线与$\overline{p_ip_{i+1}}$($\overline{p_ip_{i-1}}$)所夹的角域称为 p_i 的 $A(C)$域；$\overline{p_{i-1}p_i}$的延长线与$\overline{p_{i+1}p_i}$的延长线所夹的角域称为 p_i 的 B 域。图 2-28 中已示出凹点 p_2 的 A,B,C 域，顶点 p_4，p_8 分别落入 A,C 域，顶点 p_5，p_6，p_7 落入 B 域。利用三阶行列式的值可以判断点落入哪个域。

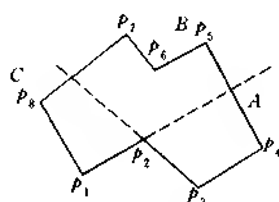


图 2-28 凹点 p_2 的 A,B,C 域

$Z_{2\text{-}4}$算法(分割多边形成凸多边形的算法)

输入 多边形 P 的顶点 $p_1(x_1,y_1)$，$p_2(x_2,y_2)$，…，$p_n(x_n,y_n)$按逆时针方向排列。

输出 凸多边形序列 W_i 及其顶点序列 p_{11}，p_{12}，…，p_{1v_1}；p_{21}，p_{22}，…，p_{2v_2}；…；p_{m1}，p_{m2}，…，p_{mv_m}，其中 $i=\overline{1,m}$。

预处理

确定多边形 P 的凸、凹顶点，并对凹点重新编号(逆时针方向排列)，设为 $Q=\{q_1(=p_k),q_2,\cdots,q_l\}$。

if $Q=\varnothing$ **then** 输出“P 为凸多边形”，终止。

else **goto** PPCP(P)

PPCP(P)

begin

步 1 $i\leftarrow 1$

步 2 **while** q_i 的 B 域中无 P 的顶点 $\wedge$ P 边$\overline{p'p''}$的两端点分别位于 q_i 的 A、C 域中 $\wedge$ $\overline{p'q_i}$、$\overline{q_ip''}$不交多边形的边 **do**

连接 p' 与 q_i，q_i 与 p''，$\overline{p'q_i}$、$\overline{q_ip''}$分割 P 成三角形 $p'q_ip''$及两个多边形 P_1 与 P_2，$Q\leftarrow Q-\{q_i\}$，重排 P_1、P_2 中凹点编号。

call PPCP(P_1)，PPCP(P_2)，直至 $Q=\varnothing$，输出凸多边形集，终止。

步 3 **while** q_i 的 B 域中无 P 的顶点 $\wedge$ q_i 的 $A(C)$域中有凹点 $q_{i+i'}$(凸点 p_m) $\wedge$ $q_i\in q_{i+i'}$的 B 域 $\wedge$ $\overline{q_iq_{i+i'}}$($\overline{q_ip_m}$)不交多边形的边 **do**

连接 q_i 与 $q_{i+i'}$(q_i 与 p_m)，Q 不变，$\overline{q_iq_{i+i'}}$($\overline{q_ip_m}$)分割 P 为 P_1 与 P_2，重排 P_1、P_2 中凹点编号。

call PPCP(P_1)，PPCP(P_2)，直至 $Q=\varnothing$，输出凸多边形集，终止。

步 4 **while** q_i 的 B 域中无 P 的顶点 $\wedge$ q_i 的 A 域中有凹点 q_{i+1} $\wedge$ $q_i\in q_{i+1}$的域 $\wedge$ $\overline{q_iq_{i+1}}$不交多边形的边 **do**

连接 q_i 与 q_{i+1}，$Q\leftarrow Q-\{q_{i+1}\}$，$\overline{q_iq_{i+1}}$分割 P 为 P_1 与凸多边形 W，重排 P_1 中凹点编号。

call PPCP(P_1)，直至 $Q=\varnothing$，输出凸多边形集，终止。

步 5　**if**　$q_j, q_{j+1}, \cdots, q_{j+j'}$ 及部分凸点 $p' \in q_i$ 的 B 域 $\wedge q_i \in q_{j+j''}(0<j''\leqslant j')$ 的 B 域 $\wedge$ $\overline{q_iq_{j+j''}}$ 不交多边形的边

then　连接 q_i 与 $q_{j+j''}$（q_i 的 B 域中无凹点时，则连接 q_i 与 p'），$Q \leftarrow Q - \{q_i, q_{j+j''}\}$，$\overline{q_iq_{j+j''}}$（$\overline{q_ip'}$）分 P 成两个多边形 P_1 与 P_2，重排 P_1、P_2 中凹点编号。

call　PPCP(P_1)，PPCP(P_2)，直至 $Q=\varnothing$，输出凸多边形集，终止。

else　**if**　$q_j, q_{j+1}, \cdots, q_{j+j'}$ 及部分凸点 $p_m \in q_i$ 的 B 域 $\wedge \overline{q_iq_{j+j''}}(0<j''\leqslant j')$（或 $\overline{q_ip_m}$）不交多边形的边

then　连接 q_i 与 $q_{j+j''}$（q_i 与 p_m），$Q \leftarrow Q - \{q_i\}$，$\overline{q_iq_{j+j''}}$（$\overline{q_ip_m}$）分 P 成两个多边形 P_1 与 P_2，重排 P_1、P_2 中凹点编号。

call　PPCP(P_1)，PPCP(P_2)，直至 $Q=\varnothing$，输出凸多边形集，终止。

end

引理 2-4　设 q_i, q_j 是多边形 P 的两个凹点（图 2-29），并且 q_i、q_j 分别落入 q_j、q_i 的 B 域，$\overline{q_iq_j}$ 不与 P 边相交，则 $\overline{q_iq_j}$ 分割 P 成两个多边形 P_1 与 P_2，并且在 P_1、P_2 中，q_i、q_j 不再是凹点。

证明　如果点 q_j 位于 $\overrightarrow{p_iq_i}$ 的左侧及 $\overrightarrow{p_{i+2}q_i}$ 的右侧（图 2-29），由定义 2-18 知，点 q_j 落入 q_i 的 B 域，连接 q_i 与 q_j，$\overline{q_iq_j}$ 划分 P 成两个多边形 P_1 与 P_2，$\angle p_iq_iq_j$ 及 $\angle q_jq_ip_{i+2}$ 均小于 π。因此在 P_1、P_2 中，点 q_i 不再是凹点。同理，点 q_j 也不是凹点。　　证毕。

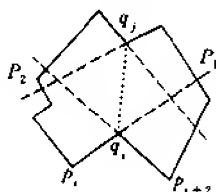


图 2-29　在 P_1、P_2 中，q_i 与 q_j 不再是凹点



图 2-30　多边形被分成三块

引理 2-5　设多边形有一个凹点 q_i，其他顶点均为凸点并且分别位于 q_i 的 C、A 域内，边 $\overline{p'p''}$ 的两个端点分别位于 C、A 域内，那么连线 $\overline{p'q_i}$ 与 $\overline{q_ip''}$ 分割多边形成一个三角形和两个凸多边形。

证明　因为 p'' 位于 q_i 的 A 域内（图 2-30），所以 $\angle p_3q_ip''<\pi$。同理，$\angle p'q_ip_1<\pi$。由于多边形的边是直线段，所以 $\angle p'q_ip''<\pi$，故结论成立。　　证毕。

定理 2-15　PPCP(P) 正确地分割多边形 P 为 $O(l)$ 个凸多边形，并且 $Z_{2\text{-}4}$ 算法的复杂性是 $\max(O(n), O(l^2))$ 次乘法，其中 n 为 P 的顶点数，l 是 P 的凹点数目。

证明　算法 $Z_{2\text{-}4}$ 首先找出多边形 P 的凸、凹顶点，然后逐个处理凹点，即把凹点变成凸点，从而达到分割多边形为凸多边形的目的。处理凹点 q_i 时，采用一种新的方法：划分 q_i 的周围区域为 A、B、C 域，依据多边形其他顶点落入不同域将作不同处理。例如，凹点 q_j 落入 q_i 的 B 域并且 q_i 落入 q_j 的 B 域，$\overline{q_iq_j}$ 不与多边形边相交，那么 $\overline{q_iq_j}$ 分割多边形成两个多边形 P_1 与 P_2，并且在 P_1、P_2 内 q_i、q_j 不再是凹点，这一点由引理 2-4 即知。同理，如果凹

点 q_j 落入 q_i 的 B 域，并且 q_i 不在 q_j 的 B 域，$\overline{q_iq_j}$不与多边形边相交，那么$\overline{q_iq_j}$分割多边形成两个多边形 P_1 与 P_2，并且在 P_1、P_2 内 q_i 不再是凹点。总之，只要连线位于某凹点的 B 域内，那么该凹点便丧失凹性而变成凸点。算法中步 4 与步 5 就是依据上述事实逐个消去凹点的。步 2 处理特殊情况，即多边形凹点 q_i 的 B 域中无多边形顶点并且 P 边$\overline{p'p''}$的两端点分别位于 q_i 的 A、C 域中，则作两条连线，由引理 2-5，多边形 P 被分割成一个三角形和两个多边形 P_1 与 P_2，并且在 P_1 与 P_2 内 q_i 不再是凹点。步 2 中情况作些改变，即 q_i 的 A、C 域中有凹点 q_{i+r}，并且引理 2-4 中的条件不成立，此时虽然原多边形被分割成两个多边形，但 q_i 与 q_{i+r}仍然是凹点，这是步 3 所做的工作。递归执行步 2 至步 5，每次递归，算法保证分割线与多边形边不相交，直至消去所有凹点，因此原多边形 P 被分割成若干个凸多边形。凸多边形的个数与凹点数目、凹点位置及凹点关联的边的方向有关(为简单起见，算法描述中省略了某些情况)。

如果每递归一次，得到两个凸多边形和一个任意多边形，并消去一个凹点，设 $T(l)$表示具有 l 个凹点的任意多边形经 PPCP(P)分割后所得凸多边形的数目，那么 $T(l)$满足下列递归关系式及初始条件：

$$\begin{cases} T(l) = T(l-1) + 2 \\ T(1) = 3 \end{cases}$$

用数学归纳法不难证明该递归关系式的解为 $T(l)=2l+1$，因此在上述假设条件下，具有 l 个凹点的任意多边形经 PPCP(P)分割后所得凸多边形的数目不超过 $2l+1=O(l)$。

利用判定点 p_{i+1}在方向线段$\overrightarrow{p_{i-1}p_i}$的左、右侧，可以确定$\angle p_{i-1}p_ip_{i+1}$小于或大于 π，从而确定点 p_i 是凸点还是凹点。由 3 阶行列式值(耗费 6 次乘法)可以判定点 p_{i+1}在$\overrightarrow{p_{i-1}p_i}$的左、右侧。因此算法中预处理需要 $6n=O(n)$次乘法。

用 12 次乘法可以确定点 p_j 是否落入 q_i 的 A、B 或 C 域中，$12(n-3)$次乘法确定 P 的 $n-3$ 个点落入 q_i 的哪个域。另外，判断两条线段是否相交需要 10 次乘法，因此耗费 $10(n-4)$次乘法可以确定线段$\overline{q_ip_j}$是否与 P 的边相交。

设 $T_1(l)$表示利用 PPCP(P)过程求解该问题所需要的乘法次数，则 $T_1(l)$满足的简化递归关系式如下：

$$T_1(l) = T_1(l-1) + 44l$$
$$T_1(1) = 44$$

可以证明，该递归关系式的解为 $T_1(l)=O(l^2)$。

算法的总耗费(以乘法次数度量)为：

$$O(n) + O(l^2) = \max(O(n), O(l^2)) = \begin{cases} O(n), & \text{如果 } l \leqslant \sqrt{2n}; \\ O(l^2), & \text{否则。} \end{cases}$$

证毕。

第3章 凸　　壳

凸壳是S计算几何中最普遍、最基本的一种结构，不仅它自身有许多特性，而且它还是构造其他几何形体的有效工具。在应用中，许多实际问题可以归结为凸壳问题。本章介绍凸壳的基本概念、计算凸壳(二维和三维)的算法及凸壳的应用。

3.1 凸壳的基本概念

设 S 是平面(E^2)中的点集，用 CH(S)表示点集 S 的凸壳，BCH(S)表示 S 的凸壳边界。

定义 3-1 设 S 是平面上的非空点集，p_1、p_2 是 S 中任意两点，如果点

$$p = tp_1 + (1-t)p_2 \tag{3-1}$$

属于 S，其中 $0 \leqslant t \leqslant 1$，则称 S 是凸集。这就是说，如果 S 中任意两点所连线段全部位于 S 之中，那么 S 是凸的。

注意，该定义可以推广到高维；此外，定义中没有规定 S 是否是连通的，是有界的还是无界的，是封闭的还是开的。显然，带有"凹部"的任何域不是凸的。

式(3-1)表示以 $p_1(t=1)$，$p_2(t=0)$ 为端点的线段，这样定义的凸集是连接 S 中任意两点的线段的集合。

凸集 S_1 与 S_2 相交，当且仅当 S_1 与 S_2 至少有一个共同点，该点既属于 S_1 又属于 S_2。

如果给定 E^2 中 k 个不同的点 $p_1, p_2, \cdots, p_k$，则点集

$$p = t_1p_1 + t_2p_2 + \cdots + t_kp_k \tag{3-2}$$

是由 $p_1, p_2, \cdots, p_k$ 生成的凸集，且 p 是 $p_1, p_2, \cdots, p_k$ 的一个凸组合，其中 t_i 是实数，$t_i \geqslant 0$，$\sum_{i=1}^{k} t_i = 1$。因此，一条线段是由其端点的所有凸组合组成，一个三角形则是由它的三个角(点)的所有凸组合组成，而三维中的四面体是由它的四个角(点)的所有凸组合组成。

定义 3-2 平面点集 S 的凸壳 CH(S)是包含 S 的最小凸集。

点集 S 的凸壳是 S 中若干点的所有凸组合的集合。对于 d 维中点集 S，CH(S)是 S 的 $d+1$(或者更少)个点的所有凸组合的集合，因此二维点集的凸壳至多是其3个点的所有凸组合的集合。每个凸组合的集合确定一个三角形，这样，平面点集 S 的凸壳是由 S 中的点确定的所有三角形的并。

点集 S 的凸壳是包含 S 的所有凸集的交，或者 CH(S)是包含 S 的所有半空间的交。二维中的半空间是半平面，它是指位于一条直线上及该线一侧的点的集合，这个概念可以推广到高维。注意，集合的凸壳是一个封闭的并且包括内部所有点的域，但在计算几何中，凸壳也理解为上述的域及其边界。

定义 3-3 平面点集 S 的凸壳边界 BCH(S)是一凸多边形，其顶点为 S 中的点。

BCH(S)是包围 S 的最小凸多边形 P,即不存在多边形 P',使得 $P \supset P' \supset S$ 成立。

BCH(S)是具有最小面积并且封闭的凸多边形 P,或者是有最小周长并封闭的凸多边形 P。

由定义 3-2 与定义 3-3,凸壳边界或者凸壳必包含凸集。可以这样来想象平面点集的凸壳,用一根橡皮圈套住 S 中的所有点,当橡皮圈收缩时形成的图形就代表了凸壳的图形,如图 3-1 所示。

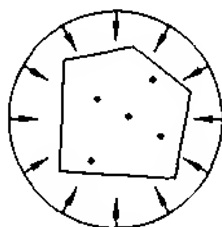


图 3-1 凸壳

平面点集 S 的凸壳或者凸壳边界是一凸多边形;反之,一个凸多边形必是一个平面点集的凸壳或凸壳边界。

利用凸壳的概念可以简化许多问题的求解,例如,点 p 到凸点集 S 的距离($=\min_{p_i \in S} d(p,p_i)$)可以用点 p 到 CH(S)的距离来代替;凸点集 S_1 与 S_2 的距离($=\min_{p_i \in S_1, q_j \in S_2} d(p_i,q_j)$)可用 CH($S_1$)与 CH($S_2$)的距离代替;$S$ 中最远两点的距离即 S 的直径以 CH(S)的直径代替;凸集 S_1 与 S_2 是否相交,可由 CH(S_1)与 CH(S_2)是否相交来决定。

BCH(S)是一凸多边形,其顶点数目 $m(k)$ 和点集中点的数目 n 有下列关系:

(1) $n \to \infty$ 时,k 维球体中均匀独立地随机分布 n 个点,其凸壳顶点数

$$m(k) = O(n^{(k-1)/(k+1)})$$

当 $k=2$(平面情况)时,$m(2)=O\left(n^{\frac{1}{3}}\right)$。

当 $k=3$(空间情况)时,$m(3)=O\left(n^{\frac{1}{2}}\right)$。

(2) $n \to \infty$ 时,点集中的点呈 k 维正态分布,则点集凸壳的顶点数

$$m(k) = O((\log n)^{(k-1)/2})$$

(3) $n \to \infty$ 时,若 k 维空间中 n 个点的分量是独立地从任何连续分布的集合中随机选取的,则其凸壳的顶点数

$$m(k) = O((\log n)^{k-1})$$

总之有 $\lim_{n \to \infty} \frac{m(k)}{n} = 0$,这表明凸壳顶点数大大少于点集中的点数。如用凸壳代替点集,那么不仅使得问题变得简单,而且可以节省存储空间。

计算平面点集 S 的凸壳就是按逆时针方向计算边界顶点。当 S 是平面上任意 n 个点的点集时,计算 CH(S)至少耗费 $O(n\log n)$时间。如果 S 是平面上任意多边形顶点序列,即 S 中的点按某种顺序排列,那么耗费线性时间便可计算 CH(S)。当 S 中的点实时增加或减少时,耗费 $O(\log^2 n)$时间可以求得 CH(S)(即保持凸壳)。

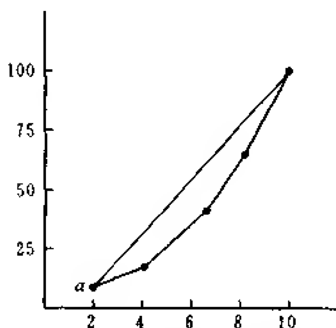


图 3-2 凸壳顶点在抛物线上

1978 年,Shamos 发现凸壳问题与分类问题之间有密切关系,利用凸壳顶点可以分类,从而由分类问题的下界可以得到凸壳问题的下界。

给定平面点集(x_i, x_i^2),如图 3-2 所示,这些点都在抛物线上,同时也在凸壳上。耗费 $n-1$ 次比较可以找到 y 坐标最小的点,即耗费 $O(n)$时间可以求得凸壳的最低点 a,

它对应于最小的 x_i。凸壳上由 a 开始按逆时针方向排列的点的 x 坐标便是分类结果，这就将凸壳顶点与分类结果联系起来了。

设 $T(n)$ 是算法 A 构造 n 个点的凸壳所需要的时间，则利用算法 A 求解分类问题耗费时间为 $T(n)+O(n)$（先构造凸壳耗费时间 $T(n)$，然后进行分类，即找点 a，耗费时间 $O(n)$），其中 $O(n)$ 表示改变凸壳为分类结果所需要的时间，也就是利用凸壳顶点进行分类所需要的时间（即上述寻找点 a 的时间）。已知分类问题的复杂性下界为 $O(n\log n)$，因此有

$$T(n) + O(n) = O(n\log n)$$

求得

$$T(n) = O(n\log n) - O(n) = O(n\log n)$$

故有下面的定理。

定理 3-1 设 S 是平面上 n 个点的点集，则计算 CH(S)至少耗费 $O(n\log n)$ 时间。

证明 设 $p_1,p_2,\cdots,p_n$ 是 S 中的 n 个点，分类这些点的 x 坐标，设 $x_i\neq x_j, i\neq j, i、j=\overline{1,n}$。对应于 x_i 构造点 (x_i,x_i^2)，即将点 p_i 移至抛物线 $y=x^2$ 上，用 q_i 表示，如图 3-3 所示。这样，CH(S)由所有点 $q_l(l=\overline{1,n})$ 组成。分类 x_i 需要 $O(n\log n)$ 时间，而由 x_i 构造点 (x_i,x_i^2) 耗费 $O(n)$ 时间（以乘法为基本操作），因此计算 CH(S)至少耗费 $O(n\log n)$ 时间。 证毕。

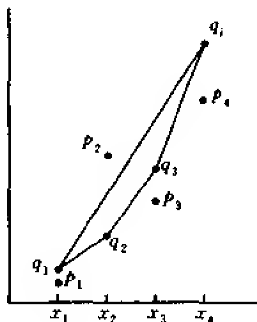


图 3-3 定理 3-1 证明的图示说明

定理 3-1 在允许乘法且分类要求 $O(n\log n)$ 时间的计算模型中均成立，它可用于维数大于 1 的空间。对于 1 维空间中的点集，凸壳是包含它们的最小区间，用线性时间可以求得该凸壳。

定理 3-2 设 S 是平面上简单多边形顶点序列，$|S|=n$，则计算 CH(S)耗费 $O(n)$ 时间。

证明 下述算法计算 CH(S)。

步 1 求 S 中点的 y 坐标最小值所对应的点，设为 p_1。

步 2 过 p_1 向右引水平线 l，计算 S 中其他各点与 p_1 的连线以及这些连线与 l 的夹角，其中最小、最大夹角的另一端点均为凸壳顶点，设为 p_a 和 $p_m(m\leqslant n, a\geqslant 2)$。这样，$p_1$，$p_a$，$p_m$ 均为凸壳顶点，其他顶点按给定多边形方式连接。

步 3 设想 $p_{a+1},p_{a+2},\cdots,p_m$ 在多边形上，然后从 $k=a+2$ 起逐个向前倒查 p_{k-1}，$p_{k-2},\cdots,p_{a+1}$ 是否在凸壳上。其检查方法是考查 p_1 和 p_k 是否在 p_{k-1} 和 p_{k-2} 连线的两侧。如果在同侧，则可暂时认为 p_{k-1} 在凸壳边界上；否则，认为 p_{k-1} 不在凸壳边界上。再查 p_1 和 p_k 是否在 p_{k-2} 和 p_{k-3} 连线的两侧。如果是同侧，则可暂时认为 p_{k-2} 在凸壳边界上；否则，p_{k-2} 不在边界上，可以删去。继续查下去。因为已确定 p_1，p_a 在凸壳边界上，对所有 k，$p_1,\cdots,p_k$ 必在 p_1 和 p_a 连线的同侧，所以上述循环检查总会成功，即位于 p_{k-b} 和 p_{k-b-1} 连线的同侧，$1\leqslant b\leqslant k-a$。该过程直到 $k=m+1$ 时终止，此时得到凸壳边界上的全部顶点。

步 1 和步 2 耗费线性时间。每个顶点至多被删去一次，所以步 3 需要 $O(n)$ 时间。总之，该算法的时间复杂性为 $O(n)$。因此定理得证。

定理 3-3 设 S 是平面上 n 个点的点集，p 是平面上一点，给定 BCH(S)的均衡分层

表示，那么在时间 $O(\log n)$ 内可以计算 $BCH(S\cup\{p\})$ 的均衡分层表示。

证明 由定理 2-1 知，耗费时间 $O(\log n)$ 可以判定 p 是否位于 BCH(S) 的内部。如果 p 在 BCH(S) 的内部，则 BCH(S) 就是 $BCH(S\cup\{p\})$，故定理成立；否则，利用二叉搜索可以计算 p 与 BCH(S) 的正切线，以求上正切线为例，过程如下：

设 q 是 BCH(S) 的顶点，如果线段 $\overline{pq}$ 与 BCH(S) 只有一个交点 q，则线段 $\overline{pq}$ 即正切线，q 为正切点。因为 S 的凸壳顶点是按逆时针方向排列的，所以只要判定与 q 相关联的两个凸壳顶点位于 $\overline{pq}$ 的哪一侧便可确定射线 $\overrightarrow{pq}$ 是进入 BCH(S) 还是离开 BCH(S)，无论这两种情况的哪一种发生，只要对 q 的前趋或后继顶点序列进行二叉搜索，耗费 $O(\log n)$ 时间就可以找到正切线及正切点。

在原均衡分层表示中再用常数次分裂和连接操作可以完成 $BCH(S\cup\{p\})$ 的均衡分层表示。因此定理成立。

定理 3-4 给定平面上两个不相交凸多边形 L、R 的均衡分层表示 $BCH(\{p_1,p_2,\cdots,p_m\})$ 和 $BCH(\{p_{m+1},\cdots,p_n\})$，并且设 $p_1\leqslant p_2\leqslant\cdots\leqslant p_m,p_{m+1}\leqslant\cdots\leqslant p_n$，其中 $\leqslant$ 是 E^2 上的词典序，则在 $O(\log n)$ 时间内可以计算 $BCH(\{p_1,p_2,\cdots,p_n\})$ 的均衡分层表示。

证明 在凸多边形 L 和 R 中分别选取顶点 r_i 和 q_j，考查与 r_i,q_j 关联的两个顶点 r_{i-1} 与 r_{i+1}，q_{j-1} 与 q_{j+1} 对方向线段 $\overrightarrow{r_iq_j}$ 的位置关系。然后再对 r_i,q_j 的前趋或后继顶点序列进行二叉搜索，耗费 $O(\log n)$ 时间可以求得上、下正切线及正切点。最后在原均衡分层表示中利用常数次分裂和连接操作便可求得 $BCH(\{p_1,p_2,\cdots,p_n\})$ 的均衡分层表示。

3.2 计算凸壳的算法(二维)

本节介绍寻求平面点集凸壳的一系列算法，包括卷包裹法、Graham(格雷厄姆)方法、分治算法、$Z_{3\text{-}1}$ 算法、$Z_{3\text{-}2}$ 算法、实时凸壳算法、增量算法及近似算法。

3.2.1 卷包裹法

卷包裹法是由 Chand 和 Kapur 于 1970 年提出的，其基本思想如下：首先过 y 坐标最小的点 p_1 画一水平直线 l，显然该点是凸壳的一个顶点。然后 l 绕 p_1 按逆时针方向旋转，碰到 S 中的第二个点 p_2 时，直线 l 改绕 p_2 按逆时针方向旋转而在 p_1 与 p_2 之间留下一线段，该线段为凸壳的一条边。继续旋转下去，最后直线 l 旋转 360°回到 p_1，便得到所要求的凸壳。

直线 l 绕点 p_i 的旋转是通过以下方法实现的：首先连接 p_i 与非凸壳顶点 p_j，$j=\overline{i+1,n}$，得到线段 $\overline{p_ip_j}$，然后求这些线段与 l(即 $\overline{p_{i-1}p_i}$) 的夹角，组成最小夹角的另一端点 p_{i+1} 即凸壳顶点。

算法(卷包裹法)

输入 平面 E^2 上 n 个点 $p_1,p_2,\cdots,p_n$ 的坐标 (x_i,y_i)，$i=\overline{1,n}$。

输出 点集 $\{p_1,p_2,\cdots,p_n\}$ 的凸壳顶点

步 1 计算 $y_1,y_2,\cdots,y_n$ 的最小值，其对应的点设为 p_1。

步 2 从 p_1 向右引一水平射线，记为 l_{p_1}。

步 3 计算$\overrightarrow{p_1p_i}$与l_{p_1}的夹角angle($\overrightarrow{p_1p_i}$,l_{p_1})及$\min\limits_i$ angle($\overrightarrow{p_1p_i}$,l_{p_1}),$i=\overline{2,n}$,设为α_1。l_{p_1}是角α_1的一条边,α_1的另一条边的端点是凸壳顶点,记为p_2。

步 4 $j\leftarrow 1, k\leftarrow 3, m\leftarrow 2$。

步 5 以$\overrightarrow{p_jp_{j+1}}$代替$\overrightarrow{p_{j-1}p_j}$($\overrightarrow{p_0p_1}=l_{p_1}$),计算$\overrightarrow{p_{j+1}p_i}$与$\overrightarrow{p_jp_{j+1}}$($i=\overline{k,n}$)的夹角angle($\overrightarrow{p_{j+1}p_i}$,$\overrightarrow{p_jp_{j+1}}$)及$\min\limits_i$ angle($\overrightarrow{p_{j+1}p_i}$,$\overrightarrow{p_jp_{j+1}}$),设为$\alpha_m$。$\alpha_m$的另一条边的端点即凸壳顶点,记为$p_k$。

步 6 $j\leftarrow j+1, k\leftarrow k+1, m\leftarrow m+1$,**goto** 步 5,直至$\alpha_m$的另一条边的端点为$p_1$。

步 7 输出凸壳顶点$p_1,p_2,\cdots,p_m$。

算法中找到的凸壳顶点p_k的编号k,可能与点集中该点的编号不一致,比如是$p_{k'}$,此时只要交换两点的编号即可。

算法中的步 1 耗费$n-1$次比较,步 2 只需要常数时间。步 3 需要计算夹角$n-1$次,然后耗费$n-2$次比较可以求得α_1。步 4 至步 6 循环$n-2$次,每次循环需要计算夹角$n-i-1$次,比较$n-i-2$次,$i=\overline{1,n-2}$。步 7 耗费常数时间。因此,算法需要计算夹角的次数为

$$n-1+\sum_{i=1}^{n-2}(n-i-1)=O(n^2)$$

比较次数为

$$n-2+\sum_{i=1}^{n-2}(n-i-2)=O(n^2)$$

故算法的时间复杂性为$O(n^2)$。

3.2.2 格雷厄姆方法

1972 年,格雷厄姆发表了一篇题为"An efficient algorithm for determining the convex hull of a finite planar set"的著名论文,这是计算几何领域中具有重要意义的早期卓越成果,文中所提出的方法称为格雷厄姆方法。

由定义 3-3 知,BCH(S)是一凸多边形,而凸多边形的各顶点必在该多边形的任意一条边的同一侧。这是格雷厄姆方法的依据。此方法步骤如下:

(1) 设凸集中y坐标最小的点为p_1,把p_1同凸集中其他各点用线段连接,并计算这些线段与水平线的夹角。然后按夹角大小及到p_1的距离进行词典式分类,得到一序列$p_1,p_2,\cdots,p_n$,依次连接这些点,便得一多边形。p_1点是凸壳边界的起点,p_2与p_n也必是凸壳顶点。$p_{n+1}=p_1$,如图 3-4 所示。

(2) 删去$p_3,p_4,\cdots,p_{n-1}$中不是凸壳顶点的点,方法如下:

begin

1. $k\leftarrow 4$
2. $j\leftarrow 2$
3. **if** p_1和p_k分别在$\overline{p_{k-j+1}p_{k-j}}$两侧 **then** 删去$p_{k-1}$,后继顶点编号减 1,
   $k\leftarrow k-1, j\leftarrow j-1$
   else p_{k-1}暂为凸壳顶点,并记录

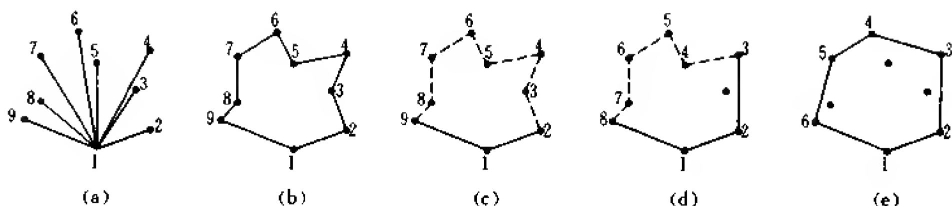


图 3-4　格雷厄姆方法的解释

(a) 依各点倾角的大小分类；(b) 按序连线成一多边形

(c) $k=4$,向前倒查,p_1 与 p_4 在 $\overline{p_{k-1}p_{k-2}}$ 两侧,所以 p_3 不在边界上,删去 p_3

(d) 原 p_4 成为 p_3,p_1 和 p_4 在 $\overline{p_{k-1}p_{k-2}}$ 同侧,p_3 暂时在边界上,继续查下去

(e) 最后得到凸壳的六个顶点

4. $j \leftarrow j+1$,　**goto** 3,直至 $j=k-1$

5. $k \leftarrow k+1$,　**goto** 2,直至 $k=n+1$

end

(3) 顺序输出凸壳顶点。

利用 5.1 节中的 some 函数

$$\text{some} = (\mathrm{d}x * \mathrm{d}y_1 - \mathrm{d}y * \mathrm{d}x_1) * (\mathrm{d}x * \mathrm{d}y_2 - \mathrm{d}y * \mathrm{d}x_2) > 0$$

可以判定点 p_1 和点 p_2 是否在线段 S 的同侧。

由于点集中有 n 个点,步骤(2)中转移到 2 的次数不超过 n,每个顶点至多被删去一次,删去顶点的个数也不可能超过 n。因此步骤(2)需要线性时间。步骤(1)要计算 $n-1$ 个夹角,并按夹角分类,计算每个夹角只需要常数时间,计算 $n-1$ 个夹角耗费线性时间,分类要时间 $O(n\log n)$。因此格雷厄姆方法的时间复杂性为 $O(n\log n)$。由定理 3-1 可以推得结论:格雷厄姆方法是求解平面点集凸壳问题的最佳算法。

3.2.3　分治算法

Preparata 和 Hong(1977)把分治技术首先应用于凸壳问题。为了说明简单起见,假设没有三个点是共线的,并且没有两个点位于一条垂线上。

把点集 S 分成两个大小近似相等的子集 S_1 和 S_2,然后分别递归地寻求 $CH(S_1)$和 $CH(S_2)$,这是两个凸多边形,设为 P_1 和 P_2,最后找 $P_1 \cup P_2$ 的凸壳。这就是分治算法求点集 S 的凸壳的基本思路,可描述如下:

步 1　把 S 中的点按 x 坐标分类

步 2　分 S 成两个子集 S_1 和 S_2,S_1 和 S_2 分别包含 $\left\lceil \frac{n}{2} \right\rceil$ 和 $\left\lfloor \frac{n}{2} \right\rfloor$ 个点。

步 3　分别计算 $P_1=CH(S_1)$和 $P_2=CH(S_2)$。

步 4　合并 $CH(S_1)$和 $CH(S_2)$,即计算 $CH(CH(S_1) \cup CH(S_2))$。

将 S 中的 n 个点按 x 坐标分类:$x_1, x_2, \cdots, x_{\lfloor \frac{n}{2} \rfloor}, x_{\lfloor \frac{n}{2} \rfloor+1}, \cdots, x_n$。$x_1, x_2, \cdots, x_{\lfloor \frac{n}{2} \rfloor}$对应的点作为 S_1,其余点作为 S_2。这种分割 S 的方法使得 P_1 与 P_2 是两个分离的凸多边形。另一种分割 S 的方法是从 S 中随机选取点轮流放入 S_1 和 S_2 中,用这种方法得到的 P_1 与

P_2是两个不一定分离的凸多边形，即$P_1 \cap P_2 \neq \varnothing$，然后寻找$CH(P_1 \cap P_2)$。可以用其他方法求子集的凸壳，能否快速合并两个子凸壳是提高分治法求凸壳效率的关键。定理3-4的证明中所示方法耗费$O(\log n)$时间可以求得$CH(P_1 \cup P_2)$，但这种方法需要预先将P_1与P_2用均衡分层表示，而构造P_1与P_2的均衡分层表示却耗费$O(n)$时间。1978年，Shamos提出完成合并工作的一种算法。现描述如下。

Procedure MERGEHULL(S)

步1 **if** $|S| \leqslant$某个小的整数 **then** 用其他方法计算$CH(S)$，终止

else **goto** 步2。

步2 分割S成两个子集S_1和S_2，$|S_1| \approx |S_2|$，$|S| = |S_1| + |S_2|$。

步3 递归地计算$CH(S_1)$和$CH(S_2)$。

步4 合并$CH(S_1)$和$CH(S_2)$形成$CH(S)$。

Procedure HULL of UNION of CONVEX POLYGONS(P_1,P_2)

步1 在P_1内部找一点p，$p \in S$。

步2 **if** p不在P_2的内部 **then** **goto** 步4

else **goto** 步3。

步3 p在P_2的内部，P_1、P_2的各顶点与p连接，并按夹角大小进行分类，得P_1和P_2顶点的一个分类表，如图3-5(a)所示，**goto** 步5。

步4 计算p与P_2的正切线，得到正切点u与v。在P_2中删去从v到u链上的顶点，保留从u到v链上的顶点，并将这些顶点及P_1顶点与p连接，再按夹角大小分类，得顶点分类表，如图3-5(b)所示。

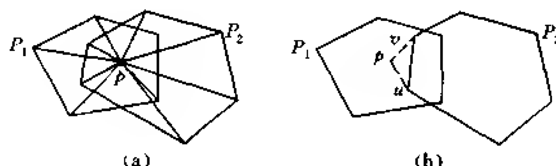


图3-5 合并P_1与P_2

步5 在顶点分类表上执行格雷厄姆方法便得到$P_1 \cup P_2$的凸壳。

上述求两个凸多边形并的壳的算法中，步1用常数时间，步2耗费$O(n)$时间。由于P_1与P_2是两个凸多边形，并且$p \in P_1 \cap P_2$，所以步3中的分类实际上是合并两个已分类的顶点表，该合并工作需要$O(n)$时间。绕P_2一周用$O(n)$时间可以判定u,v的位置，并删去从v到u的顶点链(不含v,u)，然后用$O(n)$时间合并两个顶点表，所以步4需要$O(n)$时间。步5只是使用格雷厄姆方法中的步(2)，所以步5需要$O(n)$时间。因此合并两个凸多边形耗费$O(n)$时间。

设$T(n)$表示求n个点的集合的凸壳所需要的时间，$U(n)$表示找两个凸多边形的并的壳所需要的时间，其中每个多边形有$\left\lfloor \frac{n}{2} \right\rfloor$或$\left\lceil \frac{n}{2} \right\rceil$个顶点，则有递归关系式

$$T(n) \leqslant 2T\left(\frac{n}{2}\right) + U(n) \tag{3-3}$$

由上述分析知，$U(n)=O(n)$，因此式(3-3)可以写成

$$T(n) \leqslant 2T\left(\frac{n}{2}\right) + Cn \tag{3-4}$$

其中 C 是常数。该递归关系式的解为 $T(n)=O(n\log n)$，这表明用分治法求 n 个点的集合的凸壳所需要的时间为 $O(n\log n)$。

对两个凸多边形执行"Procedure HULL of UNION of CONVEX POLYGONS"之后，还可以得到两个凸多边形的"支撑线"(即正切线)，这只要沿合并之后的凸壳边界检查，凡相邻顶点对(一点来自 P_1，另一点来自 P_2)便构成一条支撑线。

3.2.4 $Z_{3\text{-}1}$算法与 $Z_{3\text{-}2}$算法

这两个算法是周培德于 1988 年独立提出的。基本想法是先求出点集中 x，y 坐标最大、最小值，然后顺序连接最大、最小值所对应的点成四边形，该四边形划分点集为 5 个子集，不考虑位于四边形内的子集，对其他 4 个子集分别删去不是凸壳顶点的点，如图 3-6 所示。

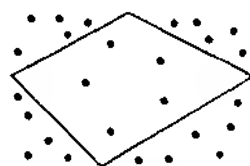


图 3-6 $Z_{3\text{-}1}$算法与 $Z_{3\text{-}2}$算法示意图

$Z_{3\text{-}1}$算法

输入 平面上 n 个点的坐标 $p_1(x_1,y_1)$，$p_2(x_2,y_2)$，…，$p_n(x_n,y_n)$，所有 n 个点存入 B:array (1..n) of point。

输出 凸壳顶点 $B_i(i=1,2,3,4)$:array (1..n_i) of point。

步 1 **if** $n=3$ **then** 三点均为凸壳顶点

else **goto** 步 2。

步 2 计算$\{x_1,x_2,\cdots,x_n\}$的最大、最小值，确定相应的点，记为 M_2，M_4(设最大、最小值只有一个)。计算$\{y_1,y_2,\cdots,y_n\}$的最大、最小值，确定相应的点，记为 M_3，M_1(设最大、最小值只有一个)。

步 3 连接 M_1，M_2，M_3 和 M_4 成四边形，$\overrightarrow{M_iM_{i+1}}(M_5=M_1)$右侧的点存入 $B_i(i=1,2,3,4)$。点 $p(x,y)$是在有向线段$\overrightarrow{p_1(x_1,y_1)p_2(x_2,y_2)}$的左侧、右侧还是在其线上分别取决于三阶行列式

$$\begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

的值是大于 0、小于 0 还是等于 0。

步 4 $i \leftarrow 1$

步 5 **if** $B_i=\varnothing$ **then** **goto** 步 7

else 将 B_i 中的点按 x 坐标分类，设分类结果为 $p_{i1}(=M_i)$，p_{i2}，…，$p_{in_i}(=M_{i+1})$。

步 6 检查 p_{i1}，p_{i2}，…，p_{in_i}中哪些点是凸壳顶点(p_{i1}和 p_{in_i}必定是凸壳顶点)。

步 6-1 依次连接 p_{i1}，p_{i2}，…，p_{in_i}，$k \leftarrow 3$。

步 6-2 **if** p_{ik}在$\overrightarrow{p_{i1}p_{i2}}$的右侧 **then** 连接 p_{i1}与 p_{ik}，删去 p_{i2}，p_{il}改为 p_{i2}，改 p_{il}为 $p_{i,l-1}(l=\overline{4,n_i-1})$，$n_i \leftarrow n_i-1$，**goto** 步 6-2；

else 连接 p_{i2}与 p_{i3}，$k \leftarrow k+1$

if $k>n_i$ then goto 步 7

else goto 步 6-3

步 6-3 if p_{ik}在$\overrightarrow{p_{i,k-2}p_{i,k-1}}$的右侧 then 连接 $p_{i,k-2}$与 p_{ik}，删去 $p_{i,k-1}$，p_{ik}改为 $p_{i,k-1}$，p_{ik}以后的点的下标减 1，$n_i \leftarrow n_i - 1$；

$k \leftarrow k-1$；

if $k>3$ then goto 步 6-3

else goto 步 6-2

else goto 步 6-4

步 6-4 if p_{ik}在$\overrightarrow{p_{i,k-2}p_{i,k-1}}$的延长线上 then 连接 $p_{i,k-1}$与 p_{ik}，$k \leftarrow k+1$；

if $k>n_i$ then goto 步 7

else goto 步 6-3

else goto 步 6-5

步 6-5 p_{ik}在$\overrightarrow{p_{i,k-2}p_{i,k-1}}$的左侧，连接 $p_{i,k-1}$与 p_{ik}，$k \leftarrow k+1$。

if $k>n_i$ then goto 步 7

else goto 步 6-3

步 7 B_i 中的 n_i 个点(包含 M_i 和 M_{i+1})即凸壳部分顶点。

步 8 if $i>4$ then goto 步 9

else $i \leftarrow i+1$ goto 步 5

步 9 $B_i(i=\overline{1,4})$中的点即凸壳顶点。

算法 $Z_{3\text{-}1}$的步 3 使得位于四边形 $M_1M_2M_3M_4$ 内部的点均不能进入 $B_i(i=\overline{1,4})$，即这些点不是凸壳顶点的候选点(但仍保留在 B 中)。对 B_i 中的点要逐点检查，按 B_i 中各点 x 坐标的顺序(B_1、B_4 中的点按递增序，B_2、B_3 中的点按递减序排序)，利用判定点在已知方向线段的右侧、左侧或其上的方法对 B_i 中的点进行筛选，其依据是凸壳顶点必在其方向边(逆时针方向为正向)的左侧(或其上)。当点 p_{ij}在某方向线段$\overrightarrow{p_{i,j-2}p_{i,j-1}}$的左侧(或其上)时，则 p_{ij}暂时保留在 B_i 中；而当点 p_{ij}在$\overrightarrow{p_{i,j-2}p_{i,j-1}}$的右侧时，则删去点 $p_{i,j-1}$。每当 k 值增加 1 或减少 1 时，p_{ik}及其之前的点就暂为凸壳顶点。每当 p_{ik}改为 $p_{i,k-1}$时，便从 B_i 中删去一个点，同时 n_i 的值减 1。经这样筛选后，B_i 中就不可能有点位于某方向线段的右侧。因此算法 $Z_{3\text{-}1}$正确地求出了平面点集的凸壳顶点。

算法 $Z_{3\text{-}1}$的步 2 需要 $2\left\lceil \frac{3}{2}n-2 \right\rceil$ 次比较操作，步 3 用线性次乘法就可以完成，步 5 分类共耗费 $\sum_{i=1}^{4} n_i \log n_i$ 次比较，步 6 至多需要 Cn_i 次判定(C 为常数)，每次判定要 9 次乘法，所以步 6 需要线性次乘法。其他步骤在常数时间内均可完成。因此算法 $Z_{3\text{-}1}$总的时间耗费为 $O(n\log n)$次比较和线性次乘法。

对算法 $Z_{3\text{-}1}$作些修改，可以得到下面的算法。

$Z_{3\text{-}2}$算法

输入、输出、步 1 至步 4 与算法 $Z_{3\text{-}1}$ 相同。

步 5 求 B_i 中各点(设 B_i 中有 n_i 个点,第一个和最后一个点分别为 M_i 和 M_{i+1})与 $\overline{M_iM_{i+1}}$ 的距离,设为 $d_{i1},d_{i2},\cdots,d_{in_i}$,其中 $d_{i1}=d_{in_i}=0$。

利用公式 $d=\dfrac{|(y_1-y_2)x+(x_2-x_1)y+(x_1y_2-x_2y_1)|}{\sqrt{(y_1-y_2)^2+(x_2-x_1)^2}}$ 可以计算点 $p(x,y)$ 与线段 $\overline{p_1(x_1,y_1)p_2(x_2,y_2)}$ 的距离。

步 6 求 $\{d_{i1},d_{i2},\cdots,d_{in_i}\}$ 的最大值(设最大值是唯一的)及其对应的点,记为 $p_i'(x_i,y_i)$。因为是求最大值,只要进行值的比较,所以可以考虑 d^2 的比较,这样可以避免开方运算。

步 7 连接 M_i 与 p_i' 和 p_i' 与 M_{i+1}。$\overrightarrow{M_ip_i'}$、$\overrightarrow{p_i'M_{i+1}}$ 右侧的点分别存入 B_{i1} 和 B_{i2}。对 B_{i1} 和 B_{i2} 分别重复步 5(此时 $\overline{M_iM_{i+1}}$ 分别改为 $\overline{M_ip_i'}$ 和 $\overline{p_i'M_{i+1}}$)与步 6 工作,并分别找到点 p_{i1}'、p_{i2}'。连接 M_i 与 p_{i1}'、p_{i1}' 与 p_i'、p_i' 与 p_{i2}'、p_{i2}' 与 M_{i+1}。递归地计算点到线段的距离。

步 8 重复步 5,6,7,直至所有 $B_{ij\cdots lm}$ 都成为空。i 由 1 增至 4,其他下标 $j,\cdots,l,m$ 取值 0、1 或 2。每次找到的点 $p_{ij\cdots l}'$ 必是凸壳的一个顶点,而全部点 $p_{ij\cdots l}'$ 便组成凸壳顶点。

设 n 个点均匀分布在同一圆周上,$n_i=\dfrac{n}{4}$,则可以证明,算法 $Z_{3\text{-}2}$ 中步 5 至步 8 所需求距离的次数为 $\sum_{i=1}^{4}(\lceil \log n_i\rceil 2^{\lfloor \log n_i\rfloor}-2^{\lfloor \log n_i\rfloor}+1)$,少于 $O(n\log n)$。总的耗费不超过 $O(n\log n)$ 次乘法和线性次比较。

3.2.5 实时凸壳算法

上述求凸壳的算法有一个共同的特点,就是在执行算法之前要求给出所有点的坐标,这种算法称为脱机算法。在许多实际问题的处理中,这一要求不能满足。一种情况是当阶段计算完成时算法才接受(或请求)一个新的输入,另一种情况是数据按某种规律(比如等时间间隔)到达,这时要求算法在下一个数据到达之前完成阶段计算。满足这两种情况要求的算法称为联机算法,而后者又称为实时算法。

脱机算法的下界也适用于联机算法,由此可推得联机算法在连续输入之间所需处理时间的下界,然后设计满足该下界要求的联机凸壳算法。

设 $T(n)$ 是联机算法求解输入规模为 n 的问题所需要的最坏情况的时间,$U(i)$ 是第 i 个输入和第 $i+1$ 个输入之间所用的时间,$L(n)$ 是求解问题所需要的时间的一个下界,则关系式

$$T(n)=\sum_{i=1}^{n-1}U(i)\geqslant L(n)$$

给出 $U(i)$ 的一个下界,即任何联机凸壳算法在相邻两点输入之间需要 $\Omega(\log n)$ 处理时间。

普雷帕拉塔(Preparata)于 1979 年设计了一个联机算法,其 $T(n)$ 为 $O(n\log n)$,$U(i)$ 为 $O(\log n)$,从而满足限界要求。

设 $C_{i-1}=\mathrm{CH}(\{p_1,p_2,\cdots,p_{i-1}\})$,时刻 i 输入点 p_i,要求在 $O(\log n)$ 时间内找到从 p_i 到 C_{i-1} 的正切线。从 p_i 看 C_{i-1},位于左(右)边的正切线称为左(右)正切线。如果 p_i 位于

C_{i-1}的内部，则不存在从 p_i 到 C_{i-1}的正切线。如果存在左、右正切线，那么用 p_i 代替两个正切点之间的顶点链便得到 $CH(\{p_1, p_2, \cdots, p_i\})$。定理 3-3 的证明过程所提供的算法也已达到限界要求。

下面再介绍一个实时凸壳算法，该算法的依据是，如果点 p' 在凸壳 CH 的内部，则射线$\overrightarrow{p'v}$(图 3-7)逆时针旋转角度 2π 之后回到初始位置；如果点 p 在 CH 的外部，射线$\overrightarrow{pv}$(v 在 CH 上)先逆时针方向旋转，到 v_l 后改为顺时针方向旋转，到 v_r 后又改为逆时针方向旋转，直至回到初始位置为止。改变旋转方向的位置恰好是正切点。

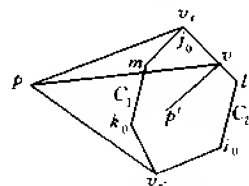


图 3-7 正切线与正切点

实时凸壳算法

步 1 **if** $|S|=3$ **then** 3 点为凸壳顶点，连接 3 点便得 CH(S)，终止。
else **goto** 步 2

步 2 任取 3 个点(不共线)组成凸壳顶点，再逐个取剩余点，执行步 3。

步 3 已知 CH(S')，$S' \subset S$，CH(S')的顶点按逆时针方向排列，简记 CH(S')为 C。
if $S-S'=\varnothing$ **then** 输出 CH(S')，终止
else 在 $S-S'$中取点 p

步 4 在 C 中任取 3 个顶点 i_0, j_0, k_0。

步 5 **if** $\angle i_0pj_0, \angle j_0pk_0, \angle k_0pi_0$ 同符号(逆时针方向为正)
then p 是 C 的内点， **goto** 步 3
else **goto** 步 6

步 6 **if** $\angle i_0pj_0$ 与$\angle j_0pk_0$ 不同符号 $\wedge i_0, j_0, k_0$ 是 C 的 3 个相邻的顶点
then j_0 是正切点， **goto** 步 7
else **if** $\angle i_0pj_0$ 与$\angle j_0pk_0$ 不同符号 $\wedge i_0, j_0, k_0$ 是 C 的 3 个不相邻的顶点，设顶点顺序为 $i_0, \cdots, l, \cdots, j_0, \cdots, m, \cdots, k_0$。
then 检查区间 i_0lj_0, lj_0m, j_0mk_0
if $\angle lpj_0$ 与$\angle j_0pm$ 不同符号(或其他角对不同符号)
then 以 l, j_0, m 代替 i_0, j_0, k_0， **goto** 步 6

步 7 重复执行步 6 一次，找到正切点 v_l 和 v_r。该正切点分裂 C 成两部分 C_1 与 C_2，如图 3-7 所示，删去内部点链 C_1，p 与 C_2(包括 v_l, v_r)构成 CH($S' \cup \{p\}$)。$S' \leftarrow S' \cup \{p\}$，**goto** 步 3。

这是一个概率算法，或称随机化算法，它所需要的执行时间显然与 3 个初始点、p 点、i_0、j_0、k_0 及 l、m 的选取有关。如果以二叉搜索方式选取 l, m，那么耗费 $O(\log n)$时间可以找到正切点 v_l 与 v_r。由于点集 S 有 n 个点，所以该算法的时间复杂性为 $O(n\log n)$。

在某些实际问题中，需要从已有的凸壳顶点中删去 1 个点 p_i，然后恢复剩余点集的凸壳，即求 CH($(S-CH(S)) \cup (CH(S)-p_i)$)。这类问题与上述问题恰好相反。从 CH($S$)中删去点 p_i 后，与 p_i 关联的 p_{i-1}、p_{i+1}之间可能要补充新的点充当凸壳顶点，如图 3-8

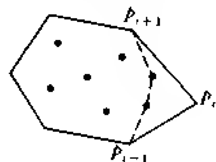


图 3-8 删去点 p_i 之后形成的新凸壳

所示。

凸壳顶点的 x 坐标最大、最小值所对应的顶点，分裂凸壳顶点链成两个子链，位于上(下)方的子链称为上(下)壳，分别记为 $U(L)$壳。向凸壳插入或从凸壳中删去一点时都用树作为数据结构。一般有两种情况：每个树结点表示一个点；仅用叶表示点，每个内点表示它的叶子的 U 壳。

设 v 表示树 T 的一个结点，$U(v)$表示存储在 v 为根的树的叶子中点的 U 壳。如果已知 v 的左、右子结点的 $U(\text{LSon}(v))$、$U(\text{RSon}(v))$，那么只需要确定两个壳的一条正切线及正切点 p_1 和 p_2。用函数 BRIDGE(U_1,U_2)产生两个 U 壳 U_1 和 U_2 的正切线。

树中每个结点带有两个信息 $Q(v)$和整数 $J(v)$，其中 $Q(v)$存储不属于 $U(\text{father}(v))$的 $U(v)$的一部分，若 v 是根，则 $Q(v)=U(v)$；$J(v)$表示 $U(v)$上左正切点的位置。

在均衡树 T 中，每个叶子结点对应一个点，每个非叶子结点对应一个桥，并附加信息 $Q(v)$和 $J(v)$，这种树称为壳树。利用壳树可以维持平面凸壳。图 3-9 是一壳树，叶中数字表示插入点集的点的顺序，内点标有两组信息：加圆括号者为 $Q(v)$，未加括号者为 $J(v)$。例如根结点有标记(11,4,1,5,12),3，其中(11,4,1,5,12)表示根对应的点集的上壳顶点链；3 表示 $U(v)$上左正切点的位置(上壳的第 3 个点)。又如根的左子结点标有(7),2，其中(7)表示不属于根(左子结点的父辈)所示上壳的一部分，即结点 7 不属于上壳；2 表示子集{11,4,3,9,1,7}的上壳左正切点的位置(即结点 4)。

图 3-9　平面点集和对应的壳树

图 3-10 与图 3-11 表示插入 p_{13}与删去 p_1 后形成的 U 壳及壳树，其中虚线表示动态修改过程及部分结果。

增加点 p_{13}由两个过程完成，第一个过程由 DESCEND(v,p)完成 p_{13}插入 T 中哪两个叶之间(形成新的叶)，第二个过程 ASCEND(v)完成树 T 中各结点信息 $Q(v)$，$J(v)$的修正。

```
Procedure  DESCEND(v,p)
begin
   if  v≠叶  then
```

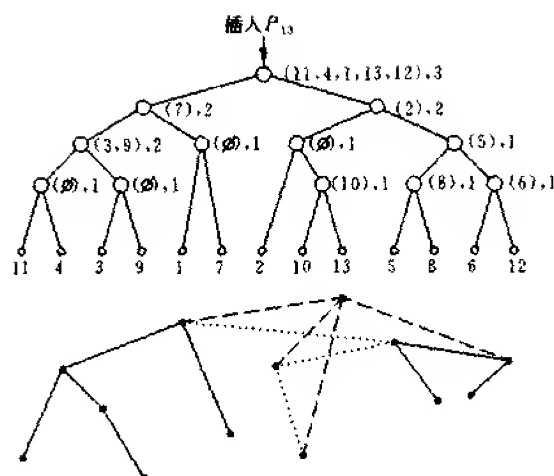


图 3-10　插入 p_{13}后形成的 U 壳及壳树

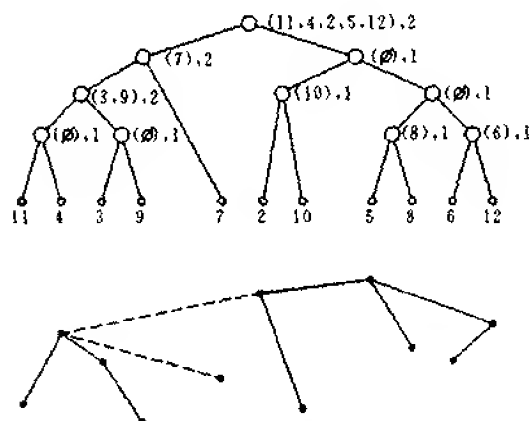


图 3-11　删去点 p_7 后形成的 U 壳及壳树

```
        begin
           (Q_L, Q_R) ← SPLIT(U(v); J(v));
           U(LSON(v)) ← SPLICE(Q_L, Q(LSON(v)));
           U(RSON(v)) ← SPLICE(Q(RSON(v)), Q_R);
           if  p.x ≤ v.x  then  v ← LSON(v)
           else  v ← RSON(v)
           DESCEND(v, p)
        end
    end
```

其中 SPLIT 是把一个串(壳顶点序列)分裂为两个子串;SPLICE 的作用是并列两个串。这两种操作各运行 $O(\log k)$时间,其中 k 为分裂之前或连接之后串的规模,因为 $k<n$,所

以在 DESCEND 中访问每个结点需要 $O(\log n)$时间。

```
Procedure  ASCEND(v)
begin
  if  v≠根  then
    begin
      (Q1,Q2,Q3,Q4;J)←BRIDGE(U(v),U(SIBLING(v)));
      Q(LSON(FATHER(v)))←Q2;
      Q(RSON(FATHER(v)))←Q3;
      U(FATHER(v))←SPLICE(Q1,Q4);
      J(FATHER(v))←J;
      ASCEND(FATHER(v))
    end
  else  Q(v)←U(v)
end
```

在 DESCEND 中访问每个结点耗费 $O(\log n)$时间，T 的深度是 $O(\log n)$，所以 DESCEND 的最坏情况下时间复杂性是 $O(\log^2 n)$。ASCEND 具有相同的时间复杂性。类似分析适用于删去一点的情况。因此，对平面中 n 个点的点集的 U 壳和 L 壳进行插入或删去，其最坏情况的时间复杂性是 $O(\log^2 n)$。

3.2.6 增量算法

由于格雷厄姆算法依赖于角度分类，而在三维情况下角度分类没有直接的对应物，故格雷厄姆算法不易推广到三维。将要介绍的增量算法及上面阐述的分治法、$Z_{3\cdot 2}$算法等则易于推广到三维。

给定 n 个点的点集 S，增量算法求 S 的凸壳的基本思想是：一次添加一个点，构造前 k 个点凸壳时用到前 $k-1$ 个点的凸壳，即每次增加一个点到已有的凸壳中去。

设 $S=\{p_1,p_2,\cdots,p_n\}$，并设 S 中任意三点都不共线，增量算法的步骤如下：

步 1　取点 p_1,p_2,p_3 围成的三角形为初始凸壳 $CH_3(\{p_1,p_2,p_3\})$

步 2　**for**　$k=4$　**to**　n　**do**

　　　$CH_k \leftarrow CH(\{CH_{k-1} \cup p_k\})$

执行步 2 时会出现两种情况：(1) 新增加的点 $p_k \in CH_{k-1}$，此时 CH_k 与 CH_{k-1}有相同的凸壳顶点；(2) 新增加的点 $p_k \notin CH_{k-1}$，则 CH_k 由 CH_{k-1}中的顶点与 p_k 组成。

按逆时针方向排列 CH_{k-1}的顶点序列，如果 p_k 在每条 BCH_{k-1}边的左侧，那么 $p_k \in CH_{k-1}$；否则，$p_k \notin CH_{k-1}$。当 $p_k \notin CH_{k-1}$时，计算 $CH(\{CH_{k-1} \cup p_k\})$，其关键是找到由 p_k 到 CH_{k-1}的正切点，下面的算法可以找到正切点。

for　$i=2$　**to**　$k-1$　**do**

　　if　(p_k 在$\overrightarrow{p_{i-1}p_i}$的左侧 $\wedge$ p_k 在$\overrightarrow{p_ip_{i+1}}$的右侧) $\vee$ (p_k 在$\overrightarrow{p_{i-1}p_i}$的右侧 $\wedge$ p_k 在$\overrightarrow{p_ip_{i+1}}$的左侧)

　　　then　p_i 是正切点

以判定点在方向线段的左、右侧为基本操作，则判定正切点算法的复杂性为$O(k)$。在最坏情况下，即n个点均为凸壳顶点，增量算法的时间复杂性为$3+4+\cdots+n=O(n^2)$。1987年，Edelsbrunner提出了改进的增量算法，其复杂性为$O(n\log n)$。

3.2.7 近似凸壳算法

有时为了提高算法的效率，减少时间开销，而降低精确性，这就是要求设计近似凸壳算法。在实际应用中，由于平面点集点的位置测量不准，即输入数据是近似的，所以设计近似凸壳算法是有意义的。

Bentley，Faust和Preparata于1982年提出了一个近似算法，其构思是从点集S中抽取某个子集S_1，然后求S_1的凸壳，把$CH(S_1)$作为$CH(S)$的近似。该算法描述如下：

步1 求S中点的x坐标最大、最小值所对应的点，设为p_{xmax}与p_{xmin}，其x值亦记为p_{xmax}与p_{xmin}。

步2 $l=\frac{p_{xmax}-p_{xmin}}{k}$，$k$为正整数，划分点集$S$所在域为$k$个长条，计算$S$中各点所在长条的编码，位于第$i$条的点构成$S_i$，$S=\bigcup_{i=1}^{k}S_i$。

步3 计算S_i中点的y坐标最大、最小值所对应的点，设为p^i_{ymax}与p^i_{ymin}，$i=\overline{1,k}$，$S^*=\{p_{xmax},p_{xmin},p^i_{ymax},p^i_{ymin}\}$。

步4 计算S^*的凸壳$CH(S^*)$，以$CH(S^*)$作为S的近似凸壳。

在步3中增加比较相邻条$S_{i-2},S_{i-1},S_i,S_{i+1},S_{i+2}$中点$y$坐标最大值，如发现两侧值大于中间值，则删去中间值小的点，对y坐标最小值也作同样处理。这样，S^*的规模将减小。

步1要求$O(n)$时间，步2耗费$O(n)$时间，步3需要的时间不超过$O(n)$。S^*至多包含$2k+4$个点，所以步4用$O(k)$时间求得S^*的凸壳。因此算法的时间复杂性为$O(n+k)$。

上述算法求得的凸壳是近似的（是真壳的子集），也就是说凸壳之外可能有S中的点，设该点为p，可以证明p离近似凸壳的距离不超过l。

3.3 计算凸壳的算法（三维）

3.2节中计算凸壳的算法，有些能推广到三维，比如，卷包裹法、分治法、$Z_{3\text{-}2}$算法及增量算法等。在介绍这些算法之前，先介绍一些基本概念。

3.3.1 基本概念

定义3-4 多面体的边界是一个平面有界凸多边形的有限集合，其中凸多边形称为多面体的侧面，两个侧面的交叫做多面体的棱，而棱与棱的交称为多面体的顶点。如果多条棱交于某顶点v，则称这些棱与顶点v关联。共享一条棱的侧面叫做邻接侧面。

多面体是由点（0维）、线段（一维）和凸多边形（二维）等几何形体围成的空间有界域。

凸多面体又叫做多胞形，或3-多胞形，凸多面体中任意两点的连线在其内部，因此凸多面体是一种特殊的多面体。凸多面体中任意两个邻接侧面所形成的二面角（多面体内

部)是凸的($<\pi$),同时与顶点 v 关联的棱之间形成的平面角的和小于 2π。

如果假设凸多面体的每个侧面均为一个平面三角形,那么将得到一种特殊的凸多面体(单纯形),下面主要介绍的由给定点集 S 求得的凸壳,就是这种类型的凸多面体。

设三维空间中给定 n 个点的点集 S,并且没有 4 个点共面。定义 3-1 的凸集、定义 3-2 的凸壳及定义 3-3 的凸壳边界也适用于三维空间,仍采用 CH(S)、BCH(S)表示点集 S 的凸壳及凸壳边界。

定义 3-5 点集 S 的凸壳边界 BCH(S)是由若干平面三角形组成,称这些三角形为 BCH(S)的侧面,三角形的边称为 BCH(S)的棱(或边),三角形的顶点称为 BCH(S)的顶点。

众所周知,BCH(S)的顶点数 v,棱数 e 和侧面数 f 满足欧拉公式 $v-e+f=2$。另外,点集 S 中的点全部位于 BCH(S)的任一侧面的同一侧。

定义 3-6 设平面三角形 abc 是 BCH(S)的一个侧面,并且三角形的 3 个顶点按逆时针方向排列,则由右手定则确定的正向(与外法线方向一致)称为在平面三角形 abc 之上,相反的方向称为在平面三角形 abc 之下。

设平面 π 上三角形 abc 的 3 个顶点 a,b,c(逆时针顺序)的坐标分别为(x_1,y_1,z_1),(x_2,y_2,z_2),(x_3,y_3,z_3),π 外点 p 的坐标为(x,y,z),则由四阶行列式

$$\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix}$$

的值为正、0、负可以确定点 p 是在三角形 abc 之上、面上及之下。如果改变三角形 abc 顶点的排列顺序,则三角形 abc 之上与之下也随之改变。

定义 3-7 如果多面体中某两个邻接侧面所形成的二面角大于 π,则称此棱为多面体的凹棱,如图 3-12(a)中的棱$\overline{p_1p_4}$。利用"判定一侧面上的点(如 p_3)是否在另一侧面三角形(如 $p_1p_2p_4$)之上"的方法可以确定该棱(比如$\overline{p_1p_4}$)是否为凹棱。

为了删去凹棱,只要改变连接方法,比如图 3-12(b),连接 p_2 与 p_3,删去$\overline{p_1p_4}$之后,便成为凸多面体。

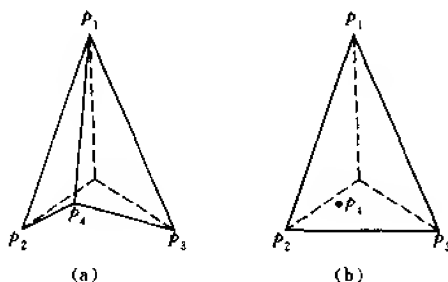


图 3-12 凹多面体与凸多面体

3.3.2 卷包裹法

卷包裹法的思想是，将点集 S 中 n 个点投影到 xy 平面上，得到点集 S'。先求出 xy 平面上 n 个投影点的凸壳，设为 CH(S')。另设线段$\overline{a'b'}$是 CH(S')的一条边界，将 a'，b'转换为 S 中的点 a，b，过$\overline{ab}$作垂直于 xy 平面的平面 π，π 绕$\overline{ab}$旋转碰到 S 中的点，设为 c，留下三角形 abc(即为 CH(S)的一个侧面)，然后 π 改绕$\overline{bc}$和$\overline{ac}$旋转碰到 S 中的点，设为 d 和 e，又留下三角形 bcd 和 ace。π 继续旋转下去，直至所有侧面三角形的边均为两个侧面三角形所共有，这样便得到点集 S 的凸壳。

平面 π 绕线段(比如$\overline{ab}$)的旋转是通过以下方法实现的：首先分别通过 S 中(除 a、b 之外)各点和线段$\overline{ab}$作平面，共计 $n-2$ 个平面。然后计算这些平面与平面 π 的夹角，夹角最小的平面设为 π_c。π_c 上只有 S 中的一个点，设为 c，连接 a，b，c 成一个三角形，该三角形即凸壳的一个侧面，这就完成了第一次旋转。其余的旋转类似进行，只是需要计算的夹角数目不断减少，这是由于对已确定为凸壳顶点的点不再作平面。

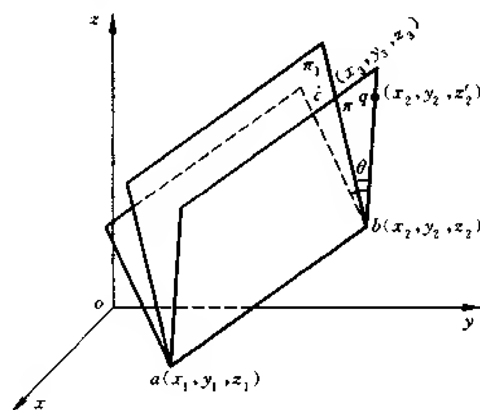


图 3-13 平面 π 与平面 π_1 之间的夹角

给定平面 π 与平面 π_1，其中平面 π 垂直于 xy 平面，如图 3-13 所示。π 过点 $a(x_1,y_1,z_1)$，$b(x_2,y_2,z_2)$，$q(x_2,y_2,z_2')$，π_1 过点 $a(x_1,y_1,z_1)$，$b(x_2,y_2,z_2)$，$c(x_3,y_3,z_3)$，则 π 与 π_1 的方程式分别为

$$Ax+By+Cz+D=0$$

其中 $A=\begin{vmatrix} y_1-y_2 & z_1-z_2' \\ 0 & z_2-z_2' \end{vmatrix}$，$B=\begin{vmatrix} x_2-x_1 & z_1-z_2' \\ 0 & z_2-z_2' \end{vmatrix}$，$C=0$，$D=-x_2\cdot\begin{vmatrix} y_1-y_2 & z_1-z_2' \\ 0 & z_2-z_2' \end{vmatrix}-y_2\cdot\begin{vmatrix} x_2-x_1 & z_1-z_2' \\ 0 & z_2-z_2' \end{vmatrix}$；

$$A_1x+B_1y+C_1z+D_1=0$$

其中 $A_1=\begin{vmatrix} y_1-y_3 & z_1-z_3 \\ y_2-y_3 & z_2-z_3 \end{vmatrix}$，$B_1=\begin{vmatrix} x_3-x_1 & z_1-z_3 \\ x_3-x_2 & z_2-z_3 \end{vmatrix}$，$C_1=\begin{vmatrix} x_1-x_3 & y_1-y_3 \\ x_2-x_3 & y_2-y_3 \end{vmatrix}$，

$$D_1=x_3\cdot\begin{vmatrix}y_3-y_1 & z_1-z_3\\ y_3-y_2 & z_2-z_3\end{vmatrix}+y_3\cdot\begin{vmatrix}x_3-x_1 & z_1-z_3\\ x_3-x_2 & z_2-z_3\end{vmatrix}+z_3\cdot\begin{vmatrix}x_3-x_1 & y_1-y_3\\ x_3-x_2 & y_2-y_3\end{vmatrix}。$$

平面 π 与平面 π_1 的夹角 θ 的余弦为

$$\cos\theta=\frac{AA_1+BB_1+CC_1}{\sqrt{A^2+B^2+C^2}\cdot\sqrt{A_1^2+B_1^2+C_1^2}}$$

算法的具体步骤如下：

步 1 计算凸壳 CH(S')，其中 S'是 S 在 xy 平面上的投影，$|S'|=n$。

步 2 在 CH(S')的边界上任取一条边，设为$\overline{p_1'p_2'}$，由 p_1'、p_2' 转换为 p_1、p_2，$\overline{p_1p_2}$是 CH(S)的一条棱。

步 3 过$\overline{p_1'p_2'}$(即$\overline{p_1p_2}$)作垂直于 xy 平面的平面 π。

步 4 分别过 S 中各点(除 p_1、p_2 外)和线段$\overline{p_1p_2}$作平面 $\pi_1,\pi_2,\cdots,\pi_{n-2}$，计算 π_i($i=\overline{1,n-2}$)与 π 的夹角 θ_i 并求最小夹角 θ 及相应平面上 S 中的点，设为 p_3。连接 p_1、p_2 与 p_3 成三角形，该三角形为 CH(S)的一个侧面，以该侧面作为转动后的 π。

步 5 分别以求得的三角形的另两条边(该三角形最初选定的边即旋转轴上的边除外)作为旋转轴，用步 4 中的方法求 CH(S)的新侧面，直至所有侧面三角形的棱均为两个侧面三角形共享。

步 1 计算 CH(S')耗费 $O(n\log n)$。步 2 和步 3 只需要常数时间。步 4 和步 5 需要 $O(n^2)$次计算夹角，求最小夹角耗费 $O(n^2)$次比较，因此算法的时间复杂性为 $O(n^2)$。

3.3.3 分治算法

分治算法求平面点集凸壳的思想可以应用于三维空间点集的凸壳问题。假设三维空间中点集 S 按均衡思想已分成两个点数大致相等的子集 S_1 与 S_2，并设已求得 CH(S_1)和 CH(S_2)，如图 3-14 所示，现要合并 CH(S_1)与 CH(S_2)成一个凸壳 CH(CH(S_1)∪CH(S_2))。为了完成这一合并工作，首先应寻找一条棱，该棱是合并后的凸壳的一条棱(图 3-14 中的线段$\overline{q_1p_1}$)；然后由该棱开始逐步构造三角形侧面，直至回到该棱。此时，这些三角形侧面及原 CH(S_1)、CH(S_2)的一部分三角形侧面组合成 S 的凸壳。

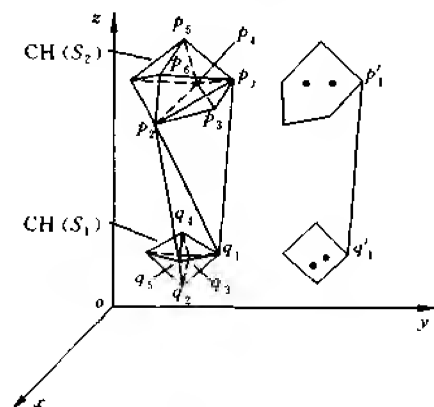


图 3-14 计算 CH(CH(S_1)∪CH(S_2))

将 S_1、S_2 分别投影到 yz 平面上，投影点集设为 S_1'、S_2'，先计算 CH(S_1')和 CH(S_2')，然后计算 CH(CH(S_1')∪CH(S_2'))，便求得线段$\overline{q_1'p_1'}$，再恢复到三维中的点 q_1 与 p_1，这样就找到了起始棱$\overline{q_1p_1}$。

由棱$\overline{q_1p_1}$开始，构造三角形侧面的过程如下：从与 p_1 关联的棱中选一条棱，比如$\overline{p_1p_2}$，连接 p_2 与 q_1，得到三角形 $p_1p_2q_1$，如果与 p_1 关联的所有其他顶点(图 3-14 中的 p_3，p_4，p_5 与 p_6)均在侧面三角形 $p_1p_2q_1$ 之下，则三角形 $p_1p_2q_1$ 便是 CH(CH(S_1)∪CH(S_2))的一个侧面；否则，更换与 p_1 关联的棱，比如选择$\overline{p_1p_3}$，再重复上述工作，直至找到棱$\overline{p_1p_i}$，使满足"与 p_1 关联的所有其他顶点均在三角形 $p_1p_iq_1$ 之下"，三角形 $p_1p_iq_1$ 是所要求的第一个侧面。然后由与 q_1 关联的棱中选一条棱，比如$\overline{q_1q_5}$，连接 q_5 与 p_2，得到三角形 $q_5q_1p_2$，如果与 q_1 关联的所有其他顶点(图 3-14 中的 q_2，q_3 与 q_4)均在侧面三角形 $q_5q_1p_2$ 之下，则三角形 $q_5q_1p_2$ 便是 CH(CH(S_1)∪CH(S_2))的第二个侧面。依此类推，直至$\overline{q_1p_1}$又成为某侧面三角形的一条棱。这时便完成了 CH(S_1)与 CH(S_2)的合并任务。

三维空间点集 S 凸壳的分治算法如下：

步 1 **if** $|S|\leqslant$某个小的正整数

　　then 用其他方法计算 CH(S)，终止。

　else **goto** 步 2

步 2 分割 S 成两个子集 S_1 和 S_2，$n_1=|S_1|\approx|S_2|=n_2$，$n=|S|=|S_1|+|S_2|=n_1+n_2$。

步 3 递归地计算 CH(S_1)和 CH(S_2)。

步 4 合并 CH(S_1)和 CH(S_2)形成 CH(S)。

Precedure MERGEHULL(CH(S_1)，CH(S_2))

步 1 将 CH(S_1)，CH(S_2)分别投影到 yz 平面上，得到点集 S_1' 和 S_2'。

步 2 计算 CH(S_1')、CH(S_2')，再计算 CH(S_1')与 CH(S_2')之间的正切线，设为$\overline{p_1'q_1'}$。

步 3 将 $p_1'q_1'$ 恢复到三维空间的线段$\overline{p_1q_1}$。

步 4 以$\overline{p_1q_1}$为起始棱，反复构造三角形侧面，得到柱形侧面 $L(S_1,S_2)$，该柱形侧面由若干平面三角形组成。$L(S_1,S_2)$将 CH(S_1)和 CH(S_2)分成两部分：柱形侧面内部 $CH_1(S_1)$，$CH_1(S_2)$和柱形侧面外部 $CH_2(S_1)$，$CH_2(S_2)$。

步 5 $CH_2(S_1)$，$CH_2(S_2)$与 $L(S_1,S_2)$构成 CH(CH(S_1)∪CH(S_2))。

合并过程中，步 1 的耗费可以不计。步 2 所需要的时间不超过 $O(n\log n)$。步 3 的时间耗费也可以不计。步 4 的耗费与柱形侧面的顶点数有关，在最坏情况下，CH(S_1)，CH(S_2)分别有 n_1-1，n_2-1 个顶点成为 $L(S_1,S_2)$的顶点，每个顶点有 3 条棱可供选择，对于其中的任一条棱又有 2 个顶点需要判断是在三角形之上或之下。这样，耗费 6 次判断是在三角形之上或之下，便可确定一个侧面三角形，因此耗费 $O(n_1)+O(n_2)$次判断，即 $O(n)$次判断可以确定 $L(S_1,S_2)$。一次判断需要 72 次乘法，所以转换成乘法，其次数也是 $O(n)$。计算 $L(S_1,S_2)$的过程中，考虑由 $L(S_1,S_2)$上边界中的相邻两条棱形成的三角形 A_i($i=\overline{1,n_1-1}$)，只要判断 CH(S_1)的各顶点位于 A_i 之上或之下，便可确定 $CH_1(S_1)$和$CH_2(S_1)$。

同理可以求得 $CH_2(S_1)$ 和 $CH_2(S_2)$。此项工作的耗费也不会超过 $O(n)$。步 5 的耗费可以不计。因此分治算法的复杂性为 $O(n\log n)$。

3.3.4 $Z_{3\text{-}3}$算法

该算法的基本思想是，先求出点集中 x,y,z 坐标最大、最小值所对应的 6 个点，设分别为 p_1,p_3,p_2,p_4,p_6 与 p_5。然后顺序连接最大、最小值所对应的点成八面体，如图 3-15 所示，S 中的点全部位于图中立方体内部，并且该八面体及其侧平面将点集 S 划分为 17 个子点集，不考虑位于八面体内的子点集。对其他 16 个子点集分别删去不是凸壳顶点的点。

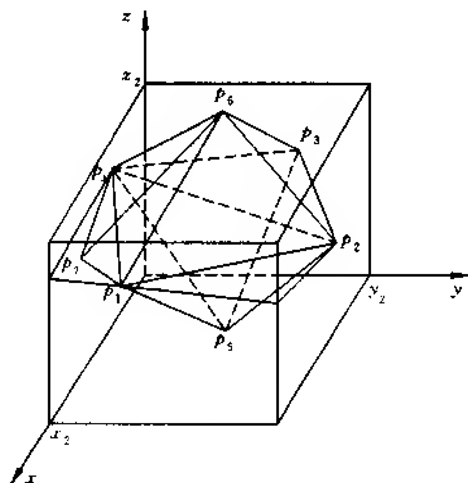


图 3-15 $Z_{3\text{-}3}$算法示意图

由 S 中点 p_i 出发引一条平行于 x 轴的射线，如果该射线与八面体不相交或有 2 个交点，则点 p_i 在八面体的外部；如果只有一个交点，则 p_i 在八面体内部。如果与八面体的棱、侧面或顶点相交，则改引平行于 y 轴或 z 轴的射线，然后再判定。如果三个方向的射线均与八面体的顶点或棱相交，则该点在八面体内部。

对于不在八面体内部的点，利用“判定点是否在侧面三角形之上”的方法可以确定点 p_i 属于哪个立体域。然后对位于同一个立体域的子点集，求该子点集中的点到该立体域侧面三角形的距离，及这些距离中的最大值。连接距离最大值所对应的点与该立体域侧面三角形 3 个顶点，便得到一个新的四面体，比如图 3-15 中的四面体 $p_7p_1p_6p_4$。对某些新增加的四面体，要检查该四面体原来的侧面三角形 3 条边是否成为凹棱，如果出现凹棱，则要改变连接方式，删去凹棱，如图 3-12 所示。

$Z_{3\text{-}3}$算法(粗略步骤)

步 1 求点集 S 中各点 x,y,z 坐标的最大、最小值所对应的点，设为 p_1,p_3,p_2,p_4,p_6 与 p_5，连接 6 点成八面体 V_8。

步 2 V_8 及部分三角形 $p_ip_jp_k(i,j,k=\overline{1,6},i\neq j\neq k)$ 所在平面将 S 划分成 17 个子集：$S_8^{内}$，$S_8^{上}(l)$，$S_8^{下}(l)$，其中 $l=\overline{1,8}$。删去八面体 V_8 内的子点集 $S_8^{内}$。落入共享侧棱的两个

三角形平面所夹的域中的子点集记为$S_8^{上}(2m)$,$S_8^{下}(2m)$,其中$m=\overline{1,4}$,记这类点为q。

步3 依次处理$S_8^{上}(2m-1)$,其中$m=\overline{1,4}$。

计算$S_8^{上}(2m-1)$中各点到侧平面三角形abc的距离,求最大距离所对应的点,设该点为p_i,连接p_i与a、p_i与b、p_i与c,得到新的四面体p_iabc,如图3-15中的$p_7p_1p_6p_4$。

步4 对于类型q的点,即$S_8^{上}(2m)$中的点,按步3中方法作出新的四面体之后要检查该新四面体的棱是否为凹棱(指与原侧面三角形形成的棱)。如果有凹棱,则改变连接方式以删去凹棱。或者将q点与共享侧棱的两个三角形的4个顶点连接,产生两个相邻(共一侧面)的四面体(此时不会生成凹棱)。

步5 删去落入新四面体内的点,新四面体外的点的子集被分成若干个更小的子集,对这些更小的子集重复步3和步4的工作,直至新的子集为空,得到上凸壳。

步6 对$S_8^{下}(2m-1)$、$S_8^{下}(2m)$(其中$m=\overline{1,4}$)中的点重复步3、步4与步5的工作,直至所有子集为空。得到下凸壳。

步7 上、下凸壳组合成CH(S)。

步1耗费$O(n)$次比较。步2利用"判定点p是否在侧面三角形之上"的方法确定点p落入哪个域,判定一个点需要72次乘法,判定n个点共需$8\times72\times n$次乘法,即$O(n)$次乘法。步3耗费$O(n)$次距离运算及$O(n)$次比较可以求得新的四面体。步4的耗费也不超过$O(n)$。步5的耗费为$\sum_{i=1}^{8}O(m_i^2)<O(n^2)$,其中$m_i$是$S_8^{上}(m)$中的点数($m=\overline{1,8}$)。步6最多重复执行$\sum_{i=1}^{8}m_i$次,最坏情况是,点位于某一子域中,其他子域为空,这时步6要循环执行n次。因此算法的最坏情况复杂性为$O(n^2)$。

3.3.5 增量算法

平面点集凸壳的增量算法可以推广到三维空间点集凸壳问题的求解。假设已求得点集$S=\{p_1,p_2,\cdots,p_k\}$的凸壳CH(S),并且点q在CH(S)之外,要求CH(CH(S)$\cup q$),如图3-16所示。

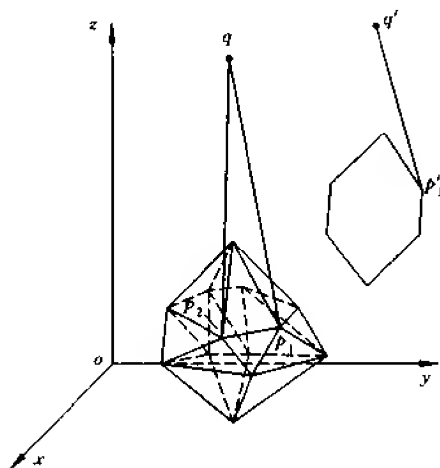


图3-16 增量算法示意图

如果把点 q 看成是一个特殊的凸壳，即由一个点组成的凸壳，那么 3.3.3 小节中介绍的分治算法可以用来计算 CH(CH(S)∪q)，只要按照该算法的合并过程中步 1 至步 5 (CH(S_2)用点 q 代替)的操作执行，便可求得 CH(CH(S)∪q)。在这个过程中，步 1 至步 3 完成寻找起始棱的任务，而这相当于寻找正切线。

计算 CH(CH(S)∪q)的另一种方法是，在 BCH(S)中随机选取一个侧面三角形，比如三角形 $p_ip_jp_k$，条件为点 q 位于该侧面三角形 $p_ip_jp_k$ 之上，并将该三角形的三个顶点与点 q 连接，得到一个四面体 $qp_ip_jp_k$。然后检查$\overline{p_ip_j}$、$\overline{p_jp_k}$与$\overline{p_kp_i}$是否为凹棱，如果不是凹棱，则 CH(S)∪$qp_ip_jp_k$ 即所求；否则，改变连接方法、从而删去凹棱。这个过程一直进行到由点 q 不能形成新的四面体为止，最后便得到 CH(CH(S)∪q)。增量算法耗费时间为 $O(n)$，计算 CH(CH(S)∪q)。

3.4 凸壳的应用

本节介绍一些实际问题，这些问题从表面上看与凸壳没有什么关系，但只要作些技巧性的转换，就变成了求凸壳问题。本节所示算法均为周培德提出。

3.4.1 确定任意多边形的凸、凹顶点

在某些应用中，需要把任意的多边形分割成若干个凸多边形。为此，如果事先能确定任意多边形各顶点是凸的还是凹的，那么这将为分割任意多边形成凸多边形提供有利条件。

如果多边形顶点序列按逆时针方向排列，并且顶点 p_{i+1} 在$\overrightarrow{p_{i-1}p_i}$的左(右)侧，那么顶点 p_i 是凸(凹)的，这种方法的时间复杂性为 $O(n)$。下面利用分割求解的思想设计了一种算法，其基本思想是：先找出任意多边形各顶点组成的点集的凸壳，该凸壳的顶点必是原多边形的部分顶点，并且这些顶点是凸的。设 p_j' 是凸壳顶点，$j=\overline{1,m}$。这 m 个点将多边形顶点序列 $p_1,p_2,\cdots,p_n$ 分成 m 个子序列，比如：

$$p_1=p_1',\underbrace{p_2,\cdots}_{1},p_i=p_2',\underbrace{p_{i+1},\cdots}_{2},\cdots,p_l=p_m',\underbrace{\cdots,p_n}_{m}$$

然后确定这些子序列组成的点集(即点链)$S_{j_k,(j+1)_k}^k$，并对点集 $S_{j_k,(j+1)_k}^k\cup\{p_j,p_{j+1}\}$分别再运用求凸壳的算法，新增的凸壳顶点必是原多边形的凹点。算法反复执行分割顶点序列与求凸壳的工作，便可确定原多边形剩余顶点的凸凹性，直至所有点集 $S_{j_k,(j+1)_k}^k$为空。

$Z_{3\text{-}4}$算法

输入 按逆时针方向排列的任意多边形顶点的坐标 $p_i(x_i,y_i)$，$i=\overline{1,n}$。

输出 点 p_i 的凸凹性("+"，"－"分别表示凸点、凹点)。

步 1 $k\leftarrow1$

步 2 利用定理 3-2 证明中的方法计算 CH($\{p_1,p_2,\cdots,p_n\}$)，设凸壳的顶点集为 $A=\{p_1',p_2',\cdots,p_m'\}$，$p_j'\leftarrow$"+"，$j=\overline{1,m}$。

　　if $m=n$ **then** 终止

else goto 步 3

步 3 确定凸壳顶点集 A 中相邻两点 p_j' 与 p_{j+1}' 之间所包含的多边形顶点组成的子集 $S_{j_k,(j+1)_k}^k, j=\overline{1,m}, p_{m+1}'=p_1'$。

if $S_{j_k,(j+1)_k}^k=\varnothing$ **then** $p_j' \leftarrow$"+", $p_{j+1}' \leftarrow$"+"

else goto 步 4

步 4 $k \leftarrow k+1, j \leftarrow 1$

步 5 $D \leftarrow$ 非空点集 $S_{j_{k-1},(j+1)_{k-1}}^{k-1} \cup \{p_j', p_{j+1}'\}$。

步 6 利用步 2 中的方法计算 CH(D)，得到凸壳的顶点集 $B'=\{B_{old}, B_{new}\}$，其中 $B_{old}=\{p_j', p_{j+1}'\}$，而 B_{new} 是新增加的凸壳顶点集。

if k 为偶数 $\wedge p \in B_{new}$ **then** $p \leftarrow$"−"

else k 为奇数 $\wedge p \in B_{new}$ **then** $p \leftarrow$"+"

步 7 求 B' 中相邻两点之间所包含的多边形顶点组成的子集 $S_{j_k,(j+1)_k}^k$。

步 8 $j \leftarrow j+1$， **goto** 步 5，直至所有非空点集 $S_{j_{k-1},(j+1)_{k-1}}^{k-1}$ 均已处理过。

步 9 重复步 4 至步 8，直至所有 $S_{j_k,(j+1)_k}^k$ 为空。

$Z_{3.4}$ 算法中的步 2，求出的顶点显然是多边形的凸点。经步 3 处理后，多边形的顶点序列被分成 m 个子序列。设 $p_i, \cdots, p_{i+a}$ 是其中的一个子序列（p_i 与 p_{i+a} 已判定是相邻的两个凸点）。$p_{i+1}, \cdots, p_{i+a-1}$ 等 $a-1$ 个顶点均在 $\overrightarrow{p_i p_{i+a}}$ 的左侧，它们的凸凹性待定。对这些点再次利用求凸壳的算法（步 6），得到凸壳顶点集 $B'=\{B_{old}, B_{new}\}$，其中 $B_{old}=\{p_i, p_{i+a}\}$，不妨设 $B_{new}=\{p_{i+1}, p_{i+4}, p_{i+a-1}\}$。$p_{i+1}, p_{i+4}, p_{i+a-1}$ 是新增加的凸壳顶点。凸壳中与点 p_{i+4} 相关联的两条新边 $\overline{p_{i+1}p_{i+4}}, \overline{p_{i+4}p_{i+a-1}}$ 的夹角和原多边形中与点 p_{i+4} 相关联的两条边的夹角满足下列关系式

$$\angle p_{i+1}p_{i+4}p_{i+a-1} \leqslant \angle p_{i+3}p_{i+4}p_{i+5} \tag{3-5}$$

又因为 $\angle p_{i+1}p_{i+4}p_{i+a-1}$ 是第二次（$k=2$）求凸壳所得的内角，该角（位于多边形内部）应大于或等于 π，所以 $\angle p_{i+3}p_{i+4}p_{i+5} \geqslant \pi$。因此顶点 p_{i+4} 是凹点。依据同样的理由，第二次求凸壳所得的新顶点均为凹点。对 m 个子序列同样考虑，求得的凸壳新顶点也是凹点。当再次循环执行步 4 至步 9 时，即第三次（$k=3$）大循环，求凸壳所得的新顶点对应的内角必等于或大于原多边形顶点对应的内角，基于类似的理由，这些顶点必是凸的。依此类推，每次大循环中求得的凸壳新顶点的凸凹性依其 k 值是奇数还是偶数而定。相邻两次大循环中所求得的凸壳新顶点的凸凹性恰好相反，这是由于式(3-5)中的不等号反向的原因。

点集 $S_{j_k,(j+1)_k}^k$ 表示了两种循环，一种是 j 由 1 增至 m，这是内循环，即处理 m 个子序列；另一种是步 4 至步 9 的大循环（$k \leftarrow k+1$），处理 m 个子序列中的子序列。每当进入一次新的大循环时，处理子序列的层次也增加 1，直至所有的 $S_{j_k,(j+1)_k}^k$ 为空。此时多边形的所有顶点均已判定是凸的还是凹的。因此，算法正确地确定了多边形顶点的凸凹性。

算法的步 3 和步 7 用性质相同的操作可以完成，该操作实际上是比较操作，因为输入时，多边形顶点的下标都是按自然数顺序排列的，即 $p_1, p_2, \cdots, p_n$。经步 2 处理后求得凸壳顶点 $p_1', p_2', \cdots, p_m'$。$p_j'(j=\overline{1,m})$ 将序列 $p_1, p_2, \cdots, p_n$ 分成 m 个子序列，只要经过 n 次比较，便可构成所有的点集 $S_{j_k,(j+1)_k}^k (k=1)$。同理，步 7 利用数目少于 n 的比较操作可以构成

点集 $S^k_{i_k,(j+1)_k}(k\geq 2)$。而步 7 最多循环执行 $n-2$ 次，故步 3 和步 7 至多需要 $O(n^2)$ 次比较。其极端情况如图 3-17 所示，步 7 要循环 8 次，第 $k+1$ 次循环比第 k 次循环少用 1 次比较，便可确定点集 $S^k_{i_k,(j+1)_k}$，这时每次循环只能确定 1 个点的凸凹性，该点划分原点序列为两个部分(子序列)，一部分为空，另一部分比原点序列少 1 个点。

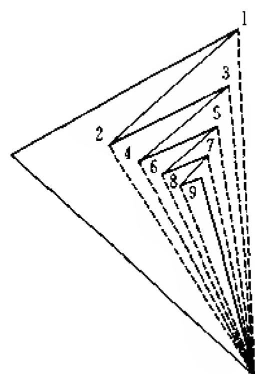


图 3-17　极端情况

步 2 和步 6 求凸壳是算法最耗费时间的步骤。在最坏情况下，步 2 求得的凸壳只包含 3 个顶点，而每次循环时，步 6 中求得的凸壳顶点集 B_{new} 只包含 1 个顶点，如图 3-17 所示。这时需要求 $n-2$ 次凸壳。由定理 3-2，计算 $\mathrm{CH}(\{p_1,p_2,\cdots,p_n\})$ 耗费 $O(n)$ 时间，算法的其他步骤耗费常数时间。因此，算法在最坏情况下的时间复杂性为 $O(n^2)$。

$Z_{3\text{-}4}$ 算法是递归结构的，并且容易转化为并行算法。

3.4.2　利用凸壳求解货郎担问题

货郎担问题是指给定 n 个点及任意两点之间的距离，求经过每个点恰好一次而且总长度最小的回路。货郎担问题是一个 NP 难问题，现已有许多近似方法求解。这里介绍的算法是依据几何中的基本运算和算法完成的。基本作法是，先求出点集的凸壳，以点集中的点与凸壳各边界的距离的最小值为标准划分点集中的点为不同类。此时注意了位于相邻两类间边界上及其附近的点与点团的处理。然后，对各类中的点集分别重复求凸壳与点分类的工作，直至所有子类为空。此时便得到一条完整的路径。该算法易于修改为并行算法。

$Z_{3\text{-}5}$ 算法

输入　n 个给定点的点集 B，包括 n 个点的坐标 $p_i(x_i,y_i)$ 及各点之间的距离 $D=(d_{ij})$。

输出　经过 n 个点的一条旅行回路 T 及此回路的长度。

步 1　求 n 个点的凸壳。设 $A=\{p_1,p_2,\cdots,p_{m_j}\}$ 是凸壳的顶点集，$C=\{\overrightarrow{p_1p_2},\overrightarrow{p_2p_3},\cdots,\overrightarrow{p_{m_j}p_1}\}$ 是凸壳的边界集，$j=1,2,\cdots$。

步 2　**if**　$m_j=n$　**then**　C 即所求回路，　**goto**　步 9

　　else　**goto**　步 3

步 3　除凸壳顶点外的 $n-m_j$ 个点分成 m_j 个类 $T^j_{i,i+1}$，$i=\overline{1,m_j}$，$T^j_{m_j,m_j+1}=T^j_{m_j,1}$。

步 3-1　计算点 $p_k\in B-A$ 至凸壳各边界的距离 $d^k_{i,i+1}$，$k=\overline{1,n-m_j}$，$i=\overline{1,m_j}$。

步 3-2　**if**　$D^k_{h,h+1}=\min\limits_i(d^k_{i,i+1})$　**then**　$T^j_{h,h+1}\leftarrow p_k$，$h=\overline{1,m_j}$

步 3-3　**if**　$T^j_{h,h+1}=\varnothing$　**then**　$T\leftarrow\overline{p_hp_{h+1}}$

步 3-4　**while**　$d^k_{i-1,i}=d^k_{i,i+1}$　**do**

　　　　if　$(p_{l_1},p_{l_2}\in T^j_{i-1,i},p'_{l_1},p'_{l_2}\in T^j_{i,i+1})\wedge$

$$\min_{p_{l_1},p_{l_2}}(d(p_k,p_{l_1})+d(p_k,p_{l_2}))<\min_{p'_{l_1},p'_{l_2}}(d(p_k,p'_{l_1})+d(p_k,p'_{l_2}))$$

then　$T^j_{i-1,i} \leftarrow p_k$　　　　/$d(p_k, p_l)$为点 p_k 与 p_l 之间的距离/

else　if　$T^j_{i-1,i} = \varnothing \vee T^j_{i,i+1} = \varnothing$

then　$(T^j_{i-1,i} \leftarrow p_k) \vee (T^j_{i,i+1} \leftarrow p_k)$

if　$T^j_{i-1,i}$与 $T^j_{i,i+1}$中只含 1 个点

then　用穷举法确定 $T^j_{i-1,i}$与 $T^j_{i,i+1}$中的路径

步 3-5　while　$d^k_{i-1,i} \approx d^k_{i,i+1} \wedge d^u_{i-1,i} \approx d^u_{i,i+1}$（点团$\{p_u\} \subset \{p_{\overline{1,n-m_j}}\}$，$p_k \in \{p_u\}$）　do

if　$\sum_u d^u_{i-1,i} < \sum_u d^u_{i,i+1}$　then　$T^j_{i-1,i} \leftarrow \{p_u\}$

else　$T^j_{i,i+1} \leftarrow \{p_u\}$

步 4　$j \leftarrow 1$

步 5　$i \leftarrow 1$

步 6　对 $T^j_{i,i+1}$重复执行步 1 与步 2，求 $T^j_{i,i+1}$中点集（包括点 p_i 与 p_{i+1}）的凸壳，删去 $\overline{p_i p_{i+1}}$。再重复执行步 3，分类 $T^j_{i,i+1}$中点集成较小的子类。

步 7　$i \leftarrow i+1$，重复执行步 6，直至 $i = m_j + 1$ 为止。

步 8　$j \leftarrow j+1$，重复执行步 5、步 6 和步 7，直至所有 $T^j_{i,i+1}$为空，便得一条完整的回路。

步 9　计算回路的长度。

算法终止。

算法的步 1、步 2 和步 3 完成求凸壳、分点集为不同子集的工作。步 6 对子点集求凸壳，并删去上一轮所求凸壳的一条边，即$\overline{p_i p_{i+1}}$。这一步保证了点集中的点的度数是 2，而且每个点都要被步 6 处理，因此算法正确地求解了货郎担问题。

由于步 3-4 与步 3-5 对位于相邻两类边界上及其附近的点与点团（位于相邻子类边界附近的 3 个及 3 个以上，并且彼此距离较小的点组成的集合称为点团）进行了优化处理，使得每条子路径的纳入都是最优化的。因此，一般来说，该算法求得的回路是较优的。

算法的步 1 要求 $O(n\log n)$次比较。步 2、步 4 和步 5 只需要常数时间。设凸壳顶点数为 m，则步 3-1 求点 $p_k \in B-A$ 至 m 条凸壳边界的距离，即计算 m 次点线距离。由于还有 $n-m$ 个点等待分类，所以步 3-1 要进行 $m(n-m)$次求点线距离的运算。步 3-2 首先从点 p_k 至 m 条边的 m 个距离中找出最小者，这要 $m-1$ 次比较，然后考虑所有 $n-m$ 个点，需要$(m-1)(n-m)$次比较。步 3-3 耗费常数时间。步 3-4 首先判断 $d^k_{i-1,i} = d^k_{i,i+1}$，如果点呈均匀分布，则需要 m 次比较，$n-m$ 个点共需 $n(n-m)$次比较；然后，验证判断条件

$$\min\left(d(p_k, p_{l_1}) + d(p_k, p_{l_2})\right) < \min\left(d(p_k, p'_{l_1}) + d(p_k, p'_{l_2})\right)$$

需要求 $2n/m$ 次两点间的距离，$(n/m)[(n/m)-1]$次比较及加法运算。该步其他操作只需常数时间。步 3-5，验证判断条件的耗费与步 3-4 耗费的数量级相同；另外，为了判断 $\sum d^u_{i-1,i} < \sum d^u_{i,i+1}$，至多需要 $2(n-m-1)$次加法和 m 次比较。

步 1 与步 3 总共需要的比较次数为 $O(n^2)$，点线距离或点点距离的计算次数为 $O(mn)$，加法次数为 $O(n^2/m^2)$。

如果假设平面上 n 个点呈均匀分布，那么步 6、步 7 和步 8 至多循环$(n-m)/m$ 次。因

此，该算法的总耗费为 $O(n^3/m)$ 次比较，$O(n^2)$ 次求距离运算及 $O(n^3/m^3)$ 次加法运算。

该算法已用于求中国 31 个省会城市的回路问题，得到一条长 15492km 的回路，比用贪心法以及改进的神经网络方法求得的回路分别缩短 1610km 和 412km，比分支限界法求得的回路长 88km。具体路线如下：

北京 $\underline{263}$ 石家庄 $\underline{394}$ 呼和浩特 $\underline{341}$ 太原 $\underline{516}$ 西安 $\underline{521}$ 银川 $\underline{346}$ 兰州 $\underline{192}$ 西宁 $\underline{1440}$ 乌鲁木齐 $\underline{1592}$ 拉萨 $\underline{1257}$ 成都 $\underline{641}$ 昆明 $\underline{434}$ 贵阳 $\underline{449}$ 南宁 $\underline{373}$ 海口 $\underline{455}$ 广州 $\underline{563}$ 长沙 $\underline{299}$ 武汉 $\underline{270}$ 南昌 $\underline{441}$ 福州 $\underline{252}$ 台北 $\underline{596}$ 杭州 $\underline{160}$ 上海 $\underline{269}$ 南京 $\underline{141}$ 合肥 $\underline{463}$ 郑州 $\underline{374}$ 济南 $\underline{271}$ 天津 $\underline{605}$ 沈阳 $\underline{281}$ 长春 $\underline{232}$ 哈尔滨 $\underline{1061}$ 北京。

对上述路线可以进行局部优化，一种方法是取路线中相邻 4 个城市构成一个四边形，在此四边形中选取其他路线与已有路线比较，取长度较短者作为通过该四城市的路线。这样可以局部改进路径长度。比如，北京—石家庄—呼和浩特—太原等城市构成一个四边形，在此四边形中的另一条路线是北京—呼和浩特—石家庄—太原，长度为 966km 比先前的路线短 32km。这样便得到一条总长度为 15460km 的路线，比最优路线只长 56km。

进一步改进的方法是在求得上述路线之后，暂时删去 $T^{i}_{n+1}=\varnothing$ 的对应点或第一次凸壳上的顶点，然后重复算法 $Z_{3\text{-}5}$ 的步骤，即重新划分点集成不同的类并求出部分路径，最后加入被删去的点。第二次（或多次）执行算法 $Z_{3\text{-}5}$ 是为了缩短部分路线的长度，从而减少总长度。

3.4.3 凸多边形直径

前面已经阐述，求平面点集的直径问题可以转化为求该点集凸壳的直径，求凸壳的直径也就是求凸多边形的直径。给定平面凸多边形 L 的顶点序列 $p_1, p_2, \cdots, p_n$，求 L 的直径，即求 n 个顶点间最远的点对。解决这个问题的一个十分简单的方法是计算 $n(n-1)/2$ 个点对的每一对点之间的距离，且选取这些距离的最大者，即要求的直径。这种方法显然需要 $O(n^2)$ 次距离运算和 $O(n^2)$ 次比较。是否存在复杂性低于 $O(n^2)$ 的算法？Preparata 和 Shamos 在"Computational Geometry: An Introduction"中提出的一种方法利用了对趾点对的概念和性质，该算法在预处理"找到所有对趾点对"之后，复杂性为 $O(n)$。现采用夹角序列方法解决了这个问题，不需要预处理，而且只需要 n 次求距离运算与 $n-1$ 次比较便可求得凸多边形 L 的直径 d。

$Z_{3\text{-}6}$ 算法

输入 凸多边形 L 的顶点序列 $p_1, p_2, \cdots, p_n$（逆时针方向排列）。

输出 凸多边形 L 的直径 $d=|\overline{p_i p_j}|$。

步 1 **for** $i=1$ **to** n **do**

$$q_{ix} \leftarrow \frac{p_{(i+1)x}+p_{ix}}{2}, q_{iy} \leftarrow \frac{p_{(i+1)y}+p_{iy}}{2}$$

步 2 $i \leftarrow 1$

步 3 $j \leftarrow i+1$

步 4 计算夹角 $\angle p_i q_j p_j$（$p_{n+1}=p_1, \cdots, p_{n+k}=p_k$；$q_{n+1}=q_1, \cdots, q_{n+k}=q_k$）

步 5 **if** $\angle p_i q_j p_j < \frac{\pi}{2}$ **then** $j \leftarrow j+1$, **goto** 步 4

else 计算距离$|\overline{p_i p_j}|$（记为 d_i）

步 6 **if** $i=n$ **then** $d \leftarrow \max(d_1, d_2, \cdots, d_n)$，输出 d，终止

else $i \leftarrow i+1$, **goto** 步 3

引理 3-1 设 D 是任意三角形 ABC 底边$\overline{BC}$的中点（见图 3-18），如果$\angle ADC < \frac{\pi}{2}$，则$|\overline{AC}| < |\overline{AB}|$。

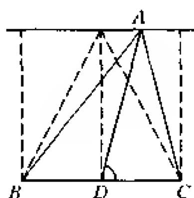


图 3-18 引理 3-1 的示意图

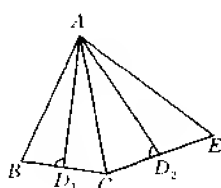


图 3-19 引理 3-2 的示意图

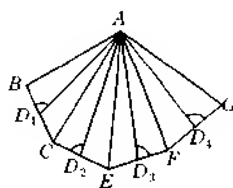


图 3-20 引理 3-3 的示意图

作如图 3-18 所示的辅助线并由简单的几何知识便可证明引理 3-1。

引理 3-2 设两个三角形 ABC、ACE 的位置如图 3-19 所示，底边$\overline{BC}$、$\overline{CE}$的中点分别为 D_1 与 D_2。如果$\angle AD_1B < \frac{\pi}{2}$并且$\angle AD_2C < \frac{\pi}{2}$，则$|\overline{AB}| < |\overline{AC}| < |\overline{AE}|$。

两次利用引理 3-1 的结果即可证明引理 3-2。

引理 3-3 设 4 个三角形 ABC、ACE、AEF 与 AFG 的位置如图 3-20 所示，底边$\overline{BC}$、$\overline{CE}$、$\overline{EF}$与$\overline{FG}$的中点分别为 D_1、D_2、D_3 与 D_4。如果$\angle AD_1B < \frac{\pi}{2}$，$\angle AD_2C < \frac{\pi}{2}$，$\angle AD_3E > \frac{\pi}{2}$并且$\angle AD_4F > \frac{\pi}{2}$，则$|\overline{AB}| < |\overline{AC}| < |\overline{AE}|$，$|\overline{AG}| < |\overline{AF}| < |\overline{AE}|$。

两次利用引理 3-2 的结果可以证明引理 3-3。

定理 3-5 给定平面上任意 n 个顶点的凸多边形 L，其顶点顺序为 $p_1, p_2, \cdots, p_n$。算法 $Z_{3\text{-}6}$正确地求出了 L 的直径，并且复杂性为：$n-1$ 次比较，n 次求距离运算及$\frac{n^2}{2}$次求夹角运算。

证明 $Z_{3\text{-}6}$算法的步 1，求出 L 的 n 条边的中点坐标，其中点为 $q_1, q_2, \cdots, q_n$。顶点 p_i 与 L 的其他顶点的连线将 L 分割为 $n-2$ 个三角形。步 4 计算夹角$\angle p_i q_j p_j$ 的大小。如果$\angle p_i q_j p_j$、$\angle p_i q_{j+1} p_{j+1}$，…，$\angle p_i q_{j'} p_{j'}$均小于$\frac{\pi}{2}$，而$\angle p_i q_{j'+1} p_{j'+1}$、$\angle p_i q_{j'+2} p_{j'+2}$，…均大于$\frac{\pi}{2}$，则由引理 3-3 知，$|\overline{p_i p_{j'+1}}|$大于 p_i 与其他顶点连线的长度。因此，步 3 至步 6 的每次循环，均求出一个顶点 p_i 至其他所有顶点连线的最大长度 d_i。步 6 的前半部工作，即当 $i=n$ 时，算法正确地求出 L 的直径。

如果 p_i 至其他所有中点连线与边的夹角序列中有两处（或多处）是由小于$\frac{\pi}{2}$至大于$\frac{\pi}{2}$的转变，则计算转变处顶点与 p_i 的连线的长度，取较长者作为顶点 p_i 至其他所有顶点

连线的最大长度 d_i。此时算法的步 5 要适当修改。

$Z_{3\text{-}6}$算法的步 1，耗费线性次四则运算。步 2、步 3 均耗费常数时间。步 3 至步 5 的循环至多需要$\frac{n}{2}$次求夹角运算与一次求距离运算。步 3 至步 6 是双重循环，外循环要执行 n 次，故步 3 至步 6 的循环需要$\frac{n^2}{2}$次求夹角运算、n 次求距离运算和 $n-1$ 次比较操作。如果以比较操作、距离运算及求夹角运算作为度量算法复杂性的基本运算，则该算法的复杂性为：$n-1$ 次比较，n 次求距离运算和$\frac{n^2}{2}$次求夹角运算。　　证毕。

利用对趾点对的方法需要求出所有的对趾点对，而每求一个对趾点对，需要求 $2\times\frac{n}{2}=n$ 个三角形的面积，即耗费 $12n$ 次乘法。对多边形的每条边都要如此处理，故求出所有对趾点对需要 $12n^2$ 次乘法。而 $Z_{3\text{-}6}$算法需要$\frac{n^2}{2}$次夹角运算，求一个夹角耗费一次除法（另需两次减法，减法耗费的时间大大少于乘、除法，故而忽略不计），所以总共需要$\frac{n^2}{2}$次除法，即大约$\frac{n^2}{2}$次乘法时间。$\frac{n^2}{2}<12n^2$，这表明 $Z_{3\text{-}6}$算法的夹角运算所耗时间小于利用对趾点对方法的预处理时间。除预处理外，两者其他开销相同，因此 $Z_{3\text{-}6}$算法优于利用对趾点对的方法。

由于距离运算需要两次乘法和一次开方，而夹角运算只要一次除法，因此距离运算的时间耗费大于夹角运算的时间耗费，故而选择距离运算为基本运算，基本运算的阶为算法时间复杂性的阶。

3.4.4 连接两个多边形成一条回路

给定两个任意简单多边形 P_1 与 P_2，它们的顶点序列分别为 $a_1,a_2,\cdots,a_n$ 与 $b_1,b_2,\cdots,b_m$。要求连接 P_1 与 P_2 成一条长度最短的回路，并保留原多边形顶点顺序。即该回路沿 P_1 与 P_2 的边（各有一条边除外）经过所有的 a_i 与 b_j（$i=\overline{1,n}$，$j=\overline{1,m}$）一次且只一次，并且回路长度最短，如图 3-21 与图 3-22 所示。解决这个问题的一种方法是计算$\overline{a_ia_{i+1}}$与$\overline{b_jb_{j+1}}$组成的四边形边长的增值（即 $|\overline{a_ib_j}|+|\overline{a_{i+1}b_{j+1}}|-|\overline{a_ia_{i+1}}|-|\overline{b_jb_{j+1}}|$的值），取增值最小者即可。这种方法需要 $8nm$ 次乘法和 nm 次比较，当 n、m 增大时，该算法的时间耗费将增加很快，因此要研究新的算法。下面介绍的算法，其基本思想是利用求凸壳顶点的方法不断删去距离较大的顶点对，从而减少计算四边形边长增值的工作。

$Z_{3\text{-}7}$算法（连接两个简单多边形成一条回路的算法）

输入　简单多边形 P_1 与 P_2，其顶点序列分别为$\{a_1,a_2,\cdots,a_n\}$与$\{b_1,b_2,\cdots,b_m\}$。

输出　经过$\{a_1,a_2,\cdots,a_n\}$与$\{b_1,b_2,\cdots,b_m\}$中各点一次且只一次的回路 C。

步 1　$i\leftarrow0,M\leftarrow\infty,k\leftarrow1,P_1(i)\leftarrow\{a_1,a_2,\cdots,a_n\},P_2(i)\leftarrow\{b_1,b_2,\cdots,b_m\}$。

步 2　分别求 $P_1(i)$的凸壳与 $P_2(i)$的凸壳，设为 $CH_1(i)$与 $CH_2(i)$。

步 3　求 $P_1(i)$与 $P_2(i)$的凸壳，设为 $CH(i)$。

步 4　确定 $CH(i)$与 $CH_1(i)$的相同点（是连续的点列），此点列的起、终点即 $CH(i)$与 $CH_1(i)$ 的切点，记为 $a_1(i)$与 $a_w(i)$（设 $a_1(i)$的 y 坐标$<a_w(i)$的 y 坐标）；确定 $CH(i)$与

$CH_2(i)$的相同点(连续的点列),此点列的起、终点即 $CH(i)$与 $CH_2(i)$的切点,记为 $b_1(i)$与 $b_{\overline{w}}(i)$(设 $b_1(i)$的 y 坐标>$b_{\overline{w}}(i)$的 y 坐标)。

步 5 与 $a_1(i)$、$a_w(i)$和 $b_1(i)$、$b_{\overline{w}}(i)$邻接的不在 $CH(i)$内的点列分别记为 $a_2(i),\cdots,a_{w-1}(i)$和 $b_2(i),\cdots,b_{\overline{w}-1}(i)$。

步 5-1 $j\leftarrow 1, j'\leftarrow \overline{w}$

步 5-2 计算 $M_k(j)\leftarrow |\overline{a_w(i)b_j(i)}|+|\overline{a_{w-1}(i)b_{j+1}(i)}|-|\overline{a_w(i)a_{w-1}(i)}|-|\overline{b_j(i)b_{j+1}(i)}|$,再计算 $|\overline{a_{w-1}(i)b_j(i)}|+|\overline{a_{w-2}(i)b_{j+1}(i)}|-|\overline{a_{w-1}(i)a_{w-2}(i)}|-|\overline{b_j(i)b_{j+1}(i)}|$的值,并与 $M_k(j)$比较,较小者保留在 $M_k(j)$。

$M_k(j')\leftarrow|\overline{a_1(i)b_{j'}(i)}|+|\overline{a_2(i)b_{j'-1}(i)}|-|\overline{a_1(i)a_2(i)}|-|\overline{b_{j'}(i)b_{j'-1}(i)}|$,再计算 $|\overline{a_2(i)b_{j'}(i)}|+|\overline{a_3(i)b_{j'-1}(i)}|-|\overline{a_2(i)a_3(i)}|-|\overline{b_{j'}(i)b_{j'-1}(i)}|$的值,并与 $M_k(j')$比较,较小者保留在 $M_k(j')$。

步 5-3 $j\leftarrow j+1, j'\leftarrow j'-1$,重复步 5-2,直至 $M_k(j)$且 $M_k(j')$首次递增或 $j=\overline{w}$且 $j'=1$,取较小者送至 M_k。

步 6 **if** $M_k<M$ **then** $M\leftarrow M_k$

步 7 从 $P_1(i)$中删去 $a_1(i)$与 $a_w(i)$之间的左侧点列(即从 $a_w(i)$起,沿包含 $CH_1(i)$中点的多边形边到达 $a_1(i)$,中间经过的点列),剩余点列记为 $P_1(i+1)$;从 $P_2(i)$中删去 $b_1(i)$与 $b_{\overline{w}}(i)$之间的左侧点列(即从 $b_{\overline{w}}(i)$起,沿包含 $CH_2(i)$中点的多边形边到达 $b_1(i)$,中间经过的点列),剩余点列记为 $P_2(i+1)$。

if $P_1(i+1)-\{a_1(i),a_w(i)\}=\varnothing$(或 $P_2(i+1)-\{b_1(i),b_{\overline{w}}(i)\}=\varnothing$)

then 保留 $a_1(i),a_w(i)$(或 $b_1(i),b_{\overline{w}}(i)$) **goto** 步 11($P_1(i+1)$记为$P_1(l)$)

else **goto** 步 8

步 8 求 $P_1(i+1)$与 $P_2(i+1)$的凸壳,设为 $CH(i+1)$。从 $CH(i+1)$中删去 $a_1(i)$与 $a_w(i)$之间的左侧点列及 $b_1(i)$与 $b_{\overline{w}}(i)$之间的左侧点列(同时从 $P_1(i+1)$,$P_2(i+1)$中删去相同点列),剩下的点列仍记为 $CH(i+1)$($P_1(i+1)$与 $P_2(i+1)$)。

步 9 从 $P_1(i+1)$中删去 $a_1(i)$与 $a_w(i)$;从 $P_2(i+1)$中删去 $b_1(i)$与 $b_{\overline{w}}(i)$。剩下的点列记为 $P_1(i+2)$与 $P_2(i+2)$。

步 10 $i\leftarrow i+2, k\leftarrow k+1$,重复步 2 至步 9,直至 $P_1(l)$或 $P_2(l)$中只有两个相邻的点或一个点或两个不相邻的点。分别转步 11、步 14 和步 16。

步 11 **if** $P_1(l)$中只有点对(a_u,a_{u+1}) **then** 保留(a_u,a_{u+1}),继续用步 2 至步 9 删去$P_2(l)$中多余的点,直至 $P_2(l)$中只有两个相邻点(b_v,b_{v+1})。此时,计算该两对点组成的四边形边长的增值,设为 m_1。

步 12 **if** $m_1<M_k$ **then** $M_k\leftarrow m_1, M\leftarrow M_k$。

步 13 由 k 确定连接 $P_1(i)$和 $P_2(i)$中对应点,并删去 $P_1(i)$和 $P_2(i)$中相应的边,便得回路 C。算法终止。

步 14 **if** $P_1(l)$中只有 1 个点 a_T **then** 保留 a_T,重复步 2 至步 9,删去 $P_2(l)$中多余的点,直至 $P_2(l)$中只有一个点 $b_{T'}$。此时,计算与 a_T,$b_{T'}$相邻的边组成的四边形边长的增值,取较小者送至 m_2。

步 15 **if** $m_2<M_k$ **then** $M_k\leftarrow m_2, M\leftarrow M_k$, **goto** 步 13。

步 16 **if** $P_1(l), P_2(l)$中只有两个不相邻的点

then 对该两点中的每个点重复步 14 和步 15。

假设多边形 P_1 与 P_2 是两个简单多边形，并且 P_1 的边与 P_2 的边不相交。另外，不妨设 P_1 顶点的 x 坐标小于 P_2 顶点的 x 坐标。$Z_{3\text{-}7}$算法的步 2、步 3 和步 4 分别找出 P_1 和 P_2 的凸壳 CH_1、CH_2 及 P_1、P_2 的凸壳 CH、切点 a_1、a_w 及 b_1、$b_{\bar{w}}$。切点分割 P_1、P_2 成左、右两侧，如图 3-21、图 3-22 所示。由于 P_1 与 P_2 是简单多边形，所以下式成立

$$\min_{\substack{a_i\in P_1\text{的左侧}\\ b_j\in P_2\text{的右侧}}} d(a_i,b_j) > \min_{\substack{a_k\in P_1\text{的右侧}\\ b_l\in P_2\text{的左侧}}} d(a_k,b_l)$$

这是 $Z_{3\text{-}7}$算法所依据的基本事实。因此 $Z_{3\text{-}7}$算法的步 7 与步 8 删去 P_1 的左侧顶点与 P_2 的右侧顶点，使得计算四边形边长增值的工作减少了。算法的步 9 与步 10 是重复删去内层的多余顶点。在删去多余顶点之前，算法的步 5 与步 6 仅计算 P_1 的右侧部分顶点与 P_2 的左侧部分顶点构成的四边形边长的增值，并保留最小的增值于 M_k 中。步 5-2 至步 5-3 的循环一般为常数次。总之，$Z_{3\text{-}7}$算法逐次删去了不具备产生最佳解的顶点而对有可能产生最佳解的顶点都进行了计算、比较，并且 M_k 中总保留着直至当前已求得的最佳解。当算法终止时，便考查了所有可能产生最佳解的顶点，并且 M_k 中保留的就是所要求的最佳解。

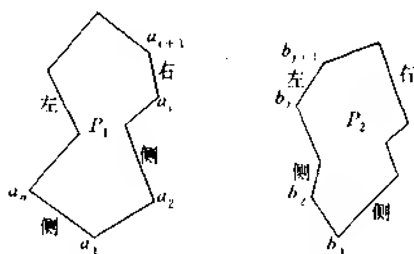


图 3-21 简单多边形 P_1 与 P_2

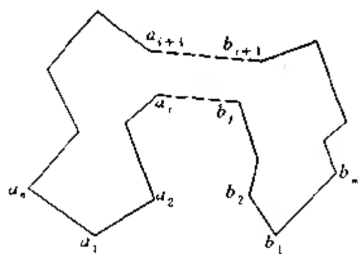


图 3-22 连接 P_1 与 P_2 成一条回路

算法的步 1、步 9 耗费常数时间。步 2 求凸壳分别需要 $O(n\log n)$, $O(m\log m)$次比较和线性次乘法。步 3 需要 $O((n+m)\log(n+m))$次比较和 $O(n+m)$次乘法。步 4 至多进行 $2(n+m)$次比较便可确定切点与相同点列。步 5 自身循环常数 C_1 次，即大约 $32C_1$ 次乘法就可求得 M_k。步 7 耗费 $\frac{n}{2}\left(\text{或}\frac{m}{2}\right)$次比较便可删去切点之间的左侧点列。步 8 进行 $O\left(\frac{1}{2}(n+m)\log\frac{(n+m)}{2}\right)$次比较和 $O\left(\frac{1}{2}(n+m)\right)$次乘法求得 $CH(i+1)$，再用 6 次乘法判定与 $a_1(i)$关联的两点的左、右侧，删去点列的时间可忽略不计。步 10 至多循环 $\max(\log n, \log m)$次。步 11、步 12 或步 14、步 15 或步 16 的循环次数也不超过 $\log n$ 或 $\log m$。总之，该算法需要的比较次数为：

$$\left[(n\log n + m\log m) + (n+m)\log(n+m) + 2(n+m) + \frac{1}{2}(n+m) + \frac{1}{2}(n+m)\log\frac{n+m}{2}\right](\max(\log n,\log m))$$

$$= O((n+m)\log(n+m)\cdot\max(\log n,\log m))$$

乘法次数为：

$$\left[(n+m)+(n+m)+32C_1+\frac{n+m}{2}+12\right]\cdot\max(\log n,\log m)$$

$$= O((n+m)\cdot\max(\log n,\log m))$$

第 4 章 Voronoi 图及其应用

就其重要性来说，Voronoi 图是仅次于凸壳的一个重要的几何结构。这是由于 Voronoi 图在求解点集或其他几何对象与距离有关的问题时起重要作用。这些问题包括，谁距离谁最近，谁距离谁最远，等等。早在 1850 年 Dirichlet 及 1908 年 Voronoi 在其论文中都讨论过 Voronoi 图的概念。

设想在一大片林区内设置 n 个火情观察塔 $p_1, p_2, \cdots, p_n$，每个观察塔 $p_i(i=\overline{1,n})$ 负责其附近林区 $V(p_i)$ 的火情发现及灭火的任务。$V(p_i)$ 由距 p_i 比距其他 $p_j(j=\overline{1,n}, j\neq i)$ 更近的树组成，$V(p_i)$ 就是关联于 p_i 的一个 Voronoi 多边形，而 Voronoi 图由所有 $V(p_i)$ 组成 $(i=\overline{1,n})$。

如果把上述 n 个观察塔换成 n 个火源，这 n 个火源同时点燃，并以相同的速度向所有方向漫延，那么燃烧熄灭处所形成的图便是 Voronoi 图。

还有许多问题都可以归结为 Voronoi 图，或利用 Voronoi 图求解。

本章介绍什么是 Voronoi 图，Voronoi 图的对偶图，Voronoi 图的性质，最近和最远意义下的 Voronoi 图，构造 Voronoi 图的算法以及 Voronoi 图的应用。

4.1 Voronoi 图的基本概念

设 p_1, p_2 是平面上两点，L 是线段 $\overline{p_1p_2}$ 的垂直平分线，L 将平面分成两部分 L_L 和 L_R，位于 L_L 内的点 p_l 具有特性：$d(p_l, p_1) < d(p_l, p_2)$，其中 $d(p_l, p_i)$ 表示 p_l 与 p_i 之间的欧几里德距离，$i=1,2$。这意味着，位于 L_L 内的点比平面其他点更接近点 p_1，换句话说，L_L 内的点是比平面上其他点更接近于 p_1 的点的轨迹，记为 $V(p_1)$，如图 4-1 所示。如果用 $H(p_1, p_2)$ 表示半平面 L_L，而 $L_R = H(p_2, p_1)$，则有 $V(p_1) = H(p_1, p_2)$，$V(p_2) = H(p_2, p_1)$。

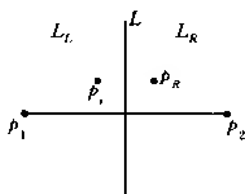


图 4-1 $V(p_1)$，$V(p_2)$ 的图示

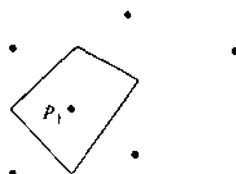


图 4-2 $n=6$ 时的一种 $V(p_1)$

给定平面上 n 个点的点集 S，$S=\{p_1, p_2, \cdots, p_n\}$。定义 $V(p_i) = \bigcap_{i\neq j} H(p_i, p_j)$，即 $V(p_i)$ 表示比其他点更接近 p_i 的点的轨迹是 $n-1$ 个半平面的交，它是一个不多于 $n-1$ 条边的凸多边形域，称为关联于 p_i 的 Voronoi 多边形或关联于 p_i 的 Voronoi 域。图 4-2 中表示

关联于 p_1 的 Voronoi 多边形，它是一个四边形，而 $n=6$。

对于 S 中的每个点都可以作一个 Voronoi 多边形，这样的 n 个 Voronoi 多边形组成的图称为 Voronoi 图，记为 Vor(S)，如图 4-3 所示。该图中的顶点和边分别称为 Voronoi 顶点和 Voronoi 边。显然，$|S|=n$ 时，Vor(S)划分平面成 n 个多边形域，每个多边形域 $V(p_i)$包含 S 中的一个点而且只包含 S 中的一个点。Vor(S)的边是 S 中某点对的垂直平分线上的一条线段或者半直线，从而为该点对所在的两个多边形域所共有。Vor(S)中有的多边形域是无界的。

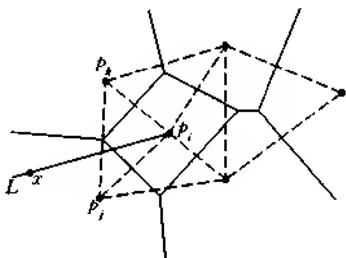


图 4-3　Voronoi 图及其对偶图

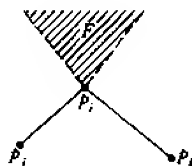


图 4-4　定理 4-1 的证明

定理 4-1　$V(p_i)$是无界的 iff $p_i \in \mathrm{BCH}(S)$。

证明　假设 $V(p_i)$是无界的而 $p_i \notin \mathrm{BCH}(S)$。由于 $V(p_i)$是一个凸多边形域，过 p_i 作一无穷射线 L，又由于 $p_i \notin \mathrm{BCH}(S)$，所以射线 L 必与 BCH(S)的某条边相交，比如与边 $\overline{p_jp_k}$相交，如图 4-3 所示。L 上离 p_i 足够远的任何点 x，x 距离 p_j 或 p_k 比距离 p_i 更近，因此 $V(p_i)$是有界的，矛盾，所以 $p_i \in \mathrm{BCH}(S)$。

反之，设 $p_i \in \mathrm{BCH}(S)$，p_j 和 p_k 也属于 BCH(S)并与 p_i 相邻(相关联)，$\overline{p_ip_k}$的垂线与 $\overline{p_ip_j}$的垂线之间形成一扇形无界域 F(图 4-4 中阴影部分)，F 中的每个点距离 p_i 比距离 S 中的任何其他点都要近，所以 $V(p_i)$是无界的。　　证毕。

Voronoi 多边形的每条边是 S 中某两点连线的垂直平分线，所有这样的两点连线构成一个图，称为 Voronoi 图的直线对偶图，记为 $D(S)$，如图 4-3 中虚线所示。对偶图的顶点是 S 中的点，边被 Voronoi 边垂直平分。

定理 4-2　n 个点的点集 S 的 Voronoi 图至多有 $2n-5$ 个顶点和 $3n-6$ 条边。

证明　Voronoi 图与其直线对偶图的边是唯一对应的，即两者边的数目相等。对偶图是一个平面图，其结点数为 n，依据欧拉公式，对偶图至多有 $3n-6$ 条边和 $2n-4$ 个面(即三角形)，所以 Voronoi 边的数目至多为 $3n-6$ 条。对偶图中只是有界面(至多 $2n-5$ 个)对偶于 Voronoi 点，也就是说，对偶图中每个三角形对偶于一个 Voronoi 点，因此，Voronoi 点至多有 $2n-5$ 个。　　证毕。

定理 4-3　每个 Voronoi 点恰好是三条 Voronoi 边的交点。

证明　设 $S=\{p_1,p_2,p_3\}$，顺序连接该三点成三角形 $p_1p_2p_3$，作三角形三条边的中垂线，它们交于一点 v_1。依据 Voronoi 图的定义，此时的 Vor(S)由三条中垂线及交点 v_1 组成。定理结论成立。

对 S 增加点时，只有两种类型情况：(1) 新增点位于$\overrightarrow{p_2p_3}$右侧、$\overrightarrow{p_1p_3}$右侧与$\overrightarrow{p_1p_2}$左侧所围成的区域内，例如图 4-5 中的点 p_4；(2)新增点位于$\overrightarrow{p_1p_3}$左侧与$\overrightarrow{p_2p_3}$右侧所围成的区域内，例如图 4-5 中的点 p_5。增加点 p_4 时，对应的 Vor(S)增加顶点 v_2，它是三角形 $p_2p_4p_3$ 三边中垂线的交点，因此定理结论成立。增加点 p_5 时，对应的 Vor(S)增加顶点 v_3 与 v_4，它们分别是三角形 $p_3p_4p_5$、三角形 $p_1p_3p_5$ 三边中垂线的交点，定理结论成立。上述过程是 Vor(S)的一种构造过程，在该过程中的任意阶段(增加一个点称为一个阶段)，定理均成立，故过程终止(不再增加点)时定理亦成立。　　证毕。

定理 4-3 表明 Voronoi 点是由 S 中三点形成的三角形的外接圆圆心，如图 4-5 中，v_1 是三角形 $p_1p_2p_3$ 的外接圆圆心，该圆记为 $C(v_1)$。显然，Voronoi 点的度数为 3，这是在条件"S 中的点没有 4 个点同在一圆周上"下得到的。

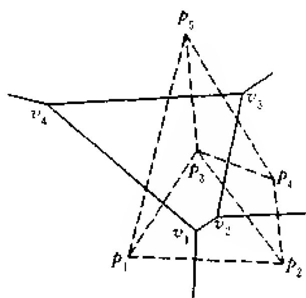


图 4-5　定理 4-3 的证明

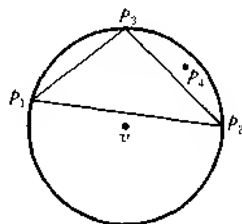


图 4-6　定理 4-4 的证明

定理 4-4　设 v 是 Vor(S)的顶点，则圆 $C(v)$内不含 S 的其他点。

证明　设 p_1、p_2、p_3 是决定圆$C(v)$的 S 中的三个点，如图 4-6 所示。如果 p_4 在圆$C(v)$内，则 p_4 与 v 的距离小于 p_1、p_2、p_3 与 v 的距离，但这与 $V(p_1)$、$V(p_2)$、$V(p_3)$的定义矛盾，故 p_4 不可能在圆 $C(v)$内，即圆 $C(v)$内不含 S 的其他的点。　　证毕。

由定理 4-4 可以知道，对于 Vor(S)中的任一个点 v，其圆 $C(v)$内均不含 S 中的点，由于这一原因，该 Vor(S)又称为最近点意义下的 Voronoi 图。

定理 4-5　S 中点 p_i 的每一个最近邻近点确定 $V(p_i)$的一条边。

证明　设 $p_i \in S$，$V(p_i)$如图 4-7 所示。由 $V(p_i)$的构成方法可知，$\overline{p_ip_j}$的垂直平分线上的一线段是 $V(p_i)$的一条边，比如图 4-7 中 e_3 是 $V(p_i)$的一条边。另外，边 e_3'、e_4' 分别过 p_j、p_k 并平行于 e_3、e_4，同理 e_1'、e_2'、e_5' 与 e_6' 分别平行于边 e_1、e_2、e_5 与 e_6。e_1'、e_2'、e_3'、e_4'、e_5' 与 e_6' 构成的多边形记为 $V'(p_i)$。如果在 $V'(p_i)$与 $V(p_i)$之间的环域内还存在 S 中的点(比如 p_l)，则由 $V(p_i)$的构成方法，必用$\overline{p_ip_l}$的垂直平分线代替 e_3，从而得到新的 $V(p_i)$，但这是不可能的，因此定理成立。

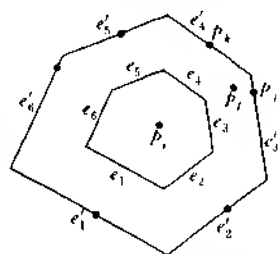


图 4-7　定理 4-5 的证明

定理 4-6　Voronoi 图的直线对偶图是 S 的一个三角剖分。

证明 定理 4-3 的证明过程所表述的方法是构造 Voronoi 图的一种方法，该方法以三角形 $p_1p_2p_3$ 为基础，不断逐个增加 S 中的点，并依据新增点所处位置的不同，决定或是增加一个 Voronoi 点或是增加两个 Voronoi 点，与此同时，或是增加一个三角形或是增加两个三角形。无论是增加一个三角形还是增加两个三角形，其周边均形成一个凸壳，如图 4-5 中虚线所示。三角形 $p_1p_2p_3$ 与新增三角形构成的图符合 Voronoi 图的直线对偶图的定义，因此定理结论成立。

定理 4-7 如果 p_i、$p_j \in S$，并且 p_j 是关于 p_i 的最近邻近点，则线段$\overline{p_ip_j}$是点集 S 三角剖分的一条边。

证明 由定理 4-3 的证明过程易见定理结论成立。

定理 4-8 如果 p_i、$p_j \in S$，并且通过 p_i 和 p_j 有一个不包含 S 中其他点的圆，那么线段$\overline{p_ip_j}$是点集 S 三角剖分的一条边，反之亦成立。

证明 由定理 4-3 和定理 4-4 可以证明。

定理 4-8 虽然不直观，但它是点集 S 三角剖分的边的一个重要特征。

定理 4-9 如果 p_i、$p_j \in S$，则线段$\overline{p_ip_j}$是 S 的三角剖分的一条边 iff 存在通过 p_i、p_j 的一个空圆：除 p_i、p_j 外不包含 S 中其他点并由该圆限界的闭圆域。

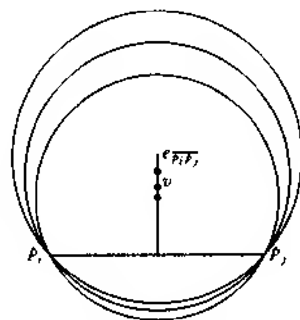


图 4-8 定理 4-9 的证明

证明 如果$\overline{p_ip_j}$是 S 的三角剖分的一条边（例如图 4-5 中$\overline{p_2p_3}$），则 $V(p_i)$和 $V(p_j)$共享边 $e \in \mathrm{Vor}(S)$，例如图 4-5 中边$\overline{v_1v_2}$。在边 e 上选择点 v（比如图 4-5 中点 v_1）为圆心，以 v 到 p_i 或 p_j 的距离为半径作圆 $C(v)$，该圆显然是不含 S 中其他点的空圆域。如若不然，比如 S 中点 p_k 在圆周上或圆内，那么 v 也会在 $V(p_k)$内，但已知 v 仅在 $V(p_i)$与 $V(p_j)$内。

反之，假设存在空圆 $C(v)$，其圆心为 v 并通过 p_i 和 p_j，要求证明$\overline{p_ip_j}$是 S 的三角剖分的一条边。因为 v 与 p_i 和 p_j 是等距离的，所以 v 在 p_i 和 p_j 的 Voronoi 域中，只要没有其他点取代"最近邻近点"。但此条件并不成立，因为该圆是空的。所以 $v \in V(p_i) \cap V(p_j)$（已定义 Voronoi 域是闭集）。由于除 p_i 与 p_j 外没有点在 $C(v)$的边界上（依据假设），故必然可以稍微移动 v 并且保持 $C(v)$内为空。具体地说，可以沿 $e_{\overline{p_ip_j}}$（p_i 与 p_j 之间的垂直平分线）移动 v，并且当圆通过 p_i 与 p_j 时，该圆保持空，如图 4-8 所示。所以 v 位于 $V(p_i)$与 $V(p_j)$之间共享的 Voronoi 边（$e_{\overline{p_ip_j}}$的子集）上。因此$\overline{p_ip_j}$是 S 的三角剖分的一条边。

证毕。

除上述定义的 Voronoi 点、Voronoi 图之外，还有一种另外定义的 Voronoi 点和 Voronoi 图，以这些点为圆心，作过 S 中三个点的圆，该圆正好包含 S 中其他全部点，这种 Voronoi 点称为最远点意义下的 Voronoi 点。这些最远点意义下的 Voronoi 点及相应的无限凸多边形组成最远点意义下的 Voronoi 图。

显然，只有通过点集 S 的凸壳边界上三个点的圆才可能把 S 中其他点包括进去，因此，n 个点的点集 S 的最远点意义下的 Voronoi 点的个数小于 n。最远点意义下的 Voronoi 图也有对偶图，该对偶图与凸壳也有共同的边界，如图 4-9 所示。

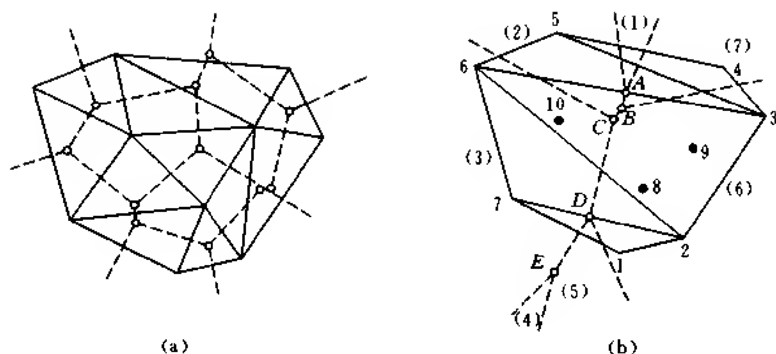


图 4-9　两种 Voronoi 图及其对偶图

图 4-9 表示 10 个点的集合 S 的两种 Voronoi 图及其对偶图，其中图 4-9(a)为最近点意义下的 Voronoi 图(虚线)及其对偶图(实线)。图 4-9(b)是最远点意义下的 Voronoi 图(虚线)及其对偶图(实线)，该 Voronoi 图具有 5 个 Voronoi 顶点 A,B,C,D 与 E，7 个无限凸多边形(1)，(2)，(3)，(4)，(5)，(6)与(7)。无限凸多边形(i)的两条半无穷直线恰好是点集 S 凸壳顶点 i 相邻的两条边的垂直平分线，$i=\overline{1,7}$。Voronoi 点 A 是三角形 1 2 7 三边垂直平分线的交点，B 是三角线 2 6 7 三边垂直平分线的交点，等等。

点集 S 的最近点意义下的 Voronoi 图，其 Voronoi 点的数目等于点集 S 三角剖分的三角形数目，而 Voronoi 多边形的数目等于点集 S 点的数目。如果点集 S 存在最远点意义下的 Voronoi 图，那么 Voronoi 点的数目等于点集凸壳的三角剖分的三角形数目，而 Voronoi 多边形的数目等于凸壳顶点数目。

4.2　构造 Voronoi 图的算法

Voronoi 图的许多应用促使研究人员成功地设计出多种构造 Voronoi 图的算法。本节介绍构造平面点集 S 的 Voronoi 图的几种算法。

4.2.1　半平面的交

利用等式 $V(p_i)=\bigcap_{i\neq j}H(p_i,p_j)$ 构造 $n-1$ 个半平面的交，得到点 p_i 的 Voronoi 多边形，然后逐点构造各点的 Voronoi 多边形便得到 S 的 Voronoi 图。算法描述如下。

利用半平面的交求 Voronoi 图的算法

步 1　按 x 坐标分类 S 中各点，设为 $p_1,p_2,\cdots,p_n$。

步 2　$i\leftarrow 1$

步 3　利用 $V(p_i)=\bigcap_{i\neq j}H(p_i,p_j)$ 求点 p_i 的 Voronoi 多边形。

步 4　$i\leftarrow i+1$，重复步 3，直到 $i>n$。

该算法步 1 的耗费为 $O(n\log n)$，步 3 求点 p_i 的 Voronoi 多边形需要 $n-i$ 次运算，步 4 至步 3 的循环耗费是

$$\sum_{i=1}^{n-1}(n-i)=O(n^2)$$

因此该算法的时间复杂性为 $O(n^2)$。

4.2.2 增量构造方法

本小节介绍构造点集 S 的 Voronoi 图的脱机增量算法和联机增量算法。

假设点集 $S=\{p_1,p_2,\cdots,p_n\}$，并设已经构造出 $k(k<n)$ 个点的 Voronoi 图 $\mathrm{Vor}(\{p_1,p_2,\cdots,p_k\})$，再增加点 p_{k+1} 之后，要求构造 Voronoi 图 $\mathrm{Vor}(\{p_1,p_2,\cdots,p_k,p_{k+1}\})$。若 p_{k+1} 位于以 Voronoi 顶点 v_i 为圆心的圆内，即 $C(v_i)$ 内，那么由定理 4-4，$\mathrm{Vor}(\{p_1,p_2,\cdots,p_k\})$ 顶点不一定是 $\mathrm{Vor}(\{p_1,p_2,\cdots,p_k,p_{k+1}\})$ 顶点。如图 4-10(a)所示，图中 $k=4$，凸壳用虚线表示，实线为 $\mathrm{Vor}(\{p_1,p_2,p_3,p_4\})$，该 Voronoi 图有两个 Voronoi 顶点 v_1 与 v_2。

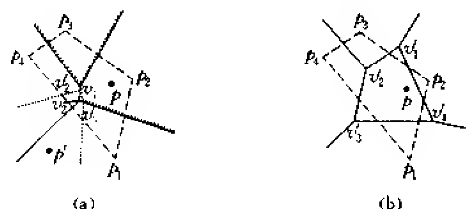


图 4-10 构造 Voronoi 图的增量算法(新增点 p 位于点集凸壳的内部或圆 $C(v_2)$ 内)

考虑增加的点 p' 在圆 $C(v_2)$ 内并位于点集凸壳之外(图 4-10(a))，此时先确定 p' 的位置是在凸壳的哪条边之右侧、或一条边的右侧、一条边的左侧(设凸壳顶点按逆时针方向排列)；然后修改相应的 Voronoi 多边形及 Voronoi 点。图 4-10(a)中，点 p' 位于凸壳边 $\overrightarrow{p_4p_1}$ 的右侧，图中点线表示 $\mathrm{Vor}(\{p_1,p_2,p_3,p_4,p'\})$，该 Voronoi 图有 3 个 Voronoi 点 v_1、v_2' 与 v_3'。显然，v_2' 与 v_3' 不是 $\mathrm{Vor}(\{p_1,p_2,p_3,p_4\})$ 的顶点；v_2 是 $\mathrm{Vor}(\{p_1,p_2,p_3,p_4\})$ 的顶点，但不是 $\mathrm{Vor}(\{p_1,p_2,p_3,p_4,p'\})$ 的顶点。

假设新增加的点 p 位于点集凸壳内(图 4-10(a))，此时应先确定点 p 所在的 Voronoi 多边形域，点 p 位于与点 p_2 相关的 Voronoi 多边形内。然后修改该 Voronoi 多边形的边与顶点，图 4-10(b)中产生 4 个新的 Voronoi 点 v_1',v_2',v_3' 与 v_4'，而原来的 Voronoi 点 v_1 与 v_2(图 4-10(a))不再是 Voronoi 点。

考虑新增加的点 p 位于凸壳的外部并且不在圆 $C(v)$ 内，其中 v 为 Voronoi 点，如图 4-11 所示。此时分两种情况讨论：(1)点 p 位于凸壳的一条边之右侧，如图 4-11(a)所示。图中点 p 在 $\overrightarrow{p_1p_2}$ 的右侧，修改点 p 所在 Voronoi 多边形的边界，得到点线所示的新增 Voronoi 多边形，并且 v_3 是新增加的 Voronoi 点。(2)点 p 位于凸壳的两条边的右侧，如图 4-11(b)所示。图中点 p 在 $\overrightarrow{p_4p_1}$ 的右侧及 $\overrightarrow{p_1p_2}$ 的右侧，修改点 p 所在 Voronoi 多边形的边界，得到点线所示的新增 Voronoi 多边形，并且 v_3、v_4 是新增加的 Voronoi 点。

1. 脱机增量算法

输入 点集 $S=\{p_1,p_2,\cdots,p_n\}$

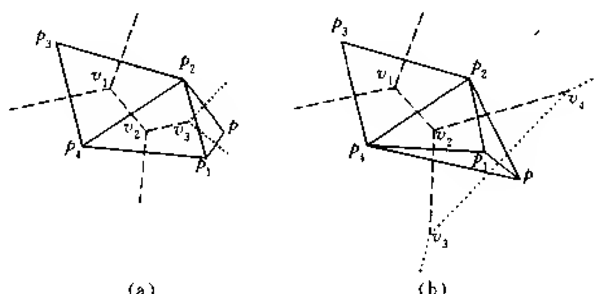


图 4-11　构造 Voronoi 图的增量算法(新增点 p 位于点集凸壳的外部及圆 $C(v)$ 的外部)

输出　点集 S 的 Voronoi 图

步 1　任取三点 p_i, p_j, p_k，并连接成三角形 $p_i p_j p_k$。

步 2　计算三角形 $p_i p_j p_k$ 外接圆圆心及半径，设为 v 和 d。

步 3　计算距离 $d(p_b, v)(b=\overline{1,n}, b \neq i, j, k)$，并将距离由小到大分类，设相应点列为 $p_1, p_2, \cdots, p_{n-3}$，$l \leftarrow 1$。

步 4　**if** $d(p_l, v) > d$ **then goto** 步 6

else goto 步 5

步 5　改取三点 p_l, p_j, p_k 或 p_i, p_l, p_k 或 p_i, p_j, p_l，并连接成三角形，或有多个点 p_1、$p_2, \cdots, p_m$ 在圆 $C(v)$ 内，则取 p_1、p_2、p_3 连接成三角形。**goto** 步 2(修改相应的下标)。

步 6　判定 p_l 在已有凸壳哪条边之右侧或两条边的右侧，如图 4-11 所示。

步 7　修改 p_l 所在 Voronoi 多边形的边界及顶点。

步 8　$l \leftarrow l+1$，**goto** 步 6，直到 $l > n-3$。

算法中步 8 至步 6 的每次循环都是在条件"p_l 不在圆 $C(v)$ 内"下执行的，其中 v 是已求得的 Voronoi 点。

算法中步 1 与步 2 只需要常数时间，步 3 要求 $n-3$ 次距离计算及 $n\log n$ 次比较。步 4 与步 5 耗费常数时间，步 5 至步 2 的循环为常数次。步 6 需要 a 次判断，其中 a 为已计算子点集凸壳的边数，每次判断要求 6 次乘法，步 7 的耗费为常数，步 8 至步 6 循环 $n-3$ 次，耗费为 $\sum_{a=3}^{n-1} a = O(n^2)$。因此算法的时间复杂性为 $O(n^2)$。

2. 联机增量算法

假设点集中的点以随机方式并间隔一段时间产生，最初只产生 1 个点 p_1，间隔 Δt 时间后产生第 2 个点 p_2，产生 p_2 后，算法立即执行，求出两个点的 Voronoi 图，即 $\overline{p_1 p_2}$ 的中垂线。随后产生第 3 个点 p_3，算法求三个点的 Voronoi 图 $\mathrm{Vor}(\{p_1, p_2, p_3\})$，依此类推。要求算法在 Δt 时间内能完成计算增加 1 个点之后的 Voronoi 图的工作。在已有 $\mathrm{Vor}(\{p_1, p_2, \cdots, p_k\})$ 的基础上，随机增加点 p_{k+1} 之后，为了计算 $\mathrm{Vor}(\{p_1, p_2, \cdots, p_k, p_{k+1}\})$，首先要判定点 p_{k+1} 位于 $\mathrm{Vor}(\{p_1, p_2, \cdots, p_k\})$ 中哪个 Voronoi 多边形域内，然后修改相应 Voronoi 多边形的边与顶点即可求得 $\mathrm{Vor}(\{p_1, p_2, \cdots, p_k, p_{k+1}\})$。

只要分别计算 p_{k+1} 与 $p_1,p_2,\cdots,p_k$ 的距离，然后求其最小距离，便可判定点 p_{k+1} 落入哪个 Voronoi 多边形域内。设 p_{k+1} 位于与 p_i 关联的 Voronoi 多边形域内，例如图 4-10(a) 中，p 位于与 p_2 关联的 Voronoi 多边形域内并且 p 在凸壳内，该多边形的边是由 p_2 分别与 p_1、p_3、p_4 的中垂线组成。修改与 p_2 关联的 Voronoi 多边形时，只要分别计算 p 与 p_1、p_2、p_3、p_4 的中垂线，见图 4-10(b)。如果 p 不在凸壳内，如图 4-10(a)中点 p'，此时 p' 位于与 p_1 关联的 Voronoi 多边形域内，该多边形的边是由 p_1 分别与 p_2、p_4 的中垂线组成，修改与 p_1 关联的 Voronoi 多边形时，只要分别计算 p' 与 p_1、p_2、p_4 的中垂线，不必计算 p' 与 p_3 的中垂线，从而节省了计算量。

构造 Voronoi 图的联机增量算法如下：

步 1　**while** 产生 p_1,p_2　**do** 作 $\overline{p_1p_2}$ 的中垂线，输出 Voronoi 图为中垂线。

步 2　**while** 产生 p_3　**do** 连接 p_1,p_2,p_3 成三角形，作三边中垂线，其交点为 Voronoi 点，从该点引出的三条中垂线构成 Voronoi 图。

步 3　$i \leftarrow 4$

步 4　**while** 产生 p_i　**do** 判定 p_i 落入哪个 Voronoi 多边形域内，修改该 Voronoi 多边形及相应 Voronoi 多边形的边与顶点。

步 5　$i \leftarrow i+1$，**goto** 步 4，直至产生点的工作终止。

算法中步 1、步 2 与步 3 均耗费常数时间。利用分别计算 p_i 至 p_1、p_2、…、p_{i-1} 的距离及 $i-2$ 次比较可以求得与 p_i 最近的点，例如 p_j，从而判定 p_i 落入与 p_j 关联的 Voronoi 多边形内，这个工作需要计算 $i-1$ 次距离及 $i-2$ 次比较。修改与 p_j 关联的 Voronoi 多边形边与顶点时，其边的数目决定了计算复杂性，由定理 4-2，n 个点的 Voronoi 图至多有 $3n-6$ 条边和 $2n-5$ 个顶点，所以每个 Voronoi 多边形边的数目为一常数，因此修改 Voronoi 多边形边与顶点耗费常数时间。设步 5 至步 4 的循环次数为 n，则算法时间复杂性为 $\sum_{i=4}^{n}(i-1)=O(n^2)$。

第 i 次执行步 4，需要 $i+2$ 次距离计算和 $i+1$ 次比较，另外还要修改相应的 Voronoi 多边形边和顶点，设修改工作所需要的时间为 C。如果用 Δt_i 表示第 i 次执行步 4 所需要的时间，那么 $\Delta t_i=$ 计算 $(i+2)$ 次距离的时间 $+(i+1)$ 次比较的时间 $+C$，这也是产生点 p_{i+3} 后，要间隔 Δt_i 时间再产生点 p_{i+4}。显然，该间隔时间是逐步增加的。

4.2.3　分治法

构造 Voronoi 图的分治算法是由 Shamos 和 Hoey (1975) 提出的，复杂性为 $O(n\log n)$。算法的基本思想是按点的 x 坐标的中值(或先按点的 x 坐标分类，然后从中间分割)分割点集 S 为 S_1 与 S_2，使 $|S_1|=|S_2|=\frac{1}{2}|S|$。如果 S_1、S_2 含点数目大于 4，则继续分割点集，直至子点集规模小于或等于 4，对每个小子点集利用 4.2.1 节或 4.2.2 节中的方法求 Voronoi 图，然后不断合并相邻子点集的 Voronoi 图，直至得到 Vor(S)。算法描述如下：

步 1　划分 S 为规模近似相等的两个子集 S_1 和 S_2。

步 2　递归地构造 Vor(S_1)和 Vor(S_2)。

步 3 构造折线 B，分开 S_1 和 S_2，并使 $d(a,v)=d(b,v)$，其中 $a\in S_1$，$b\in S_2$，v 为折线上的点。

步 4 删去位于 B 左侧的 Vor(S_2)的所有边及位于 B 右侧的 Vor(S_1)的所有边，得到集合 S 的 Voronoi 图 Vor(S)。

组成折线 B 的每条线段是 S_1 与 S_2 中某两点连线的垂直平分线。假设已知 CH(S_1)和 CH(S_2)，在线性时间内可以求得 CH(S_1)和 CH(S_2)的正切线。设$\overline{p_1p_2}$为所求的正切线，$p_1\in S_1$，$p_2\in S_2$。作$\overline{p_1p_2}$的垂直平分线。设想由上向下沿该垂直平分线下移的 z 点遇到 Vor(S_2)或者 Vor(S_1)的一条边，比如遇到 Vor(S_2)的一条边，如图 4-12 所示。图中点 p_7 和 p_{14}分别属于 $S_1=\{p_1,p_2,\cdots,p_8\}$和 $S_2=\{p_9,p_{10},\cdots,p_{16}\}$，$\overline{p_{14}p_7}$的向下垂直平分线首先与 Vor($S_2$)的边相交，即与$\overline{p_{11}p_{14}}$的垂直平分线相交，交点为 q_1，留下折线 B 的第一段(半直线)。点 q_1 是$\overline{p_{14}p_{11}}$的垂直平分线与$\overline{p_{14}p_7}$的垂直平分线的交点，因此 q_1 是三角形 $p_{14}p_7p_{11}$ 的外接圆的圆心，所以下一段折线为$\overline{p_7p_{11}}$的垂直平分线上的一条线段，此时寻找$\overline{p_7p_{11}}$的垂直平分线与 p_7 关联的 Voronoi 多边形的哪条边相交，图中示出$\overline{p_7p_{11}}$的垂直平分线与$\overline{p_7p_6}$的垂直平分线相交，交点为 q_2。$\overline{q_2q_3}$为$\overline{p_{11}p_6}$的垂直平分线上的一条线段，即寻找$\overline{p_{11}p_6}$的垂直平分线与 p_{11}关联的 Voronoi 多边形的哪条边相交，如果与 p_{11}关联的 Voronoi 多边形的多条边相交，则以比 q_2 的 y 坐标小的点作为 B 的下一个顶点 q_3，q_3 为$\overline{p_{11}p_6}$的垂直平分线与$\overline{p_{11}p_{10}}$的垂直平分线的交点。同理，$\overline{q_3q_4}$为$\overline{p_6p_{10}}$的垂直平分线上的一条线段(q_4 为$\overline{p_6p_{10}}$的垂直平分线与$\overline{p_6p_5}$的垂直平分线的交点)，$\overline{q_4q_5}$为$\overline{p_{10}p_5}$的垂直平分线上的一条线段(q_5 为$\overline{p_5p_{10}}$的垂直平分线与$\overline{p_5p_8}$的垂直平分线的交点)，$\overline{q_5q_6}$为$\overline{p_{10}p_8}$的垂直平分线上的一条线段(q_6 为$\overline{p_{10}p_8}$的垂直平分线与$\overline{p_{10}p_{12}}$的垂直平分线的交点)，$\overline{q_6q_7}$为$\overline{p_8p_{12}}$的垂直平分线上的一条线段(q_7 为$\overline{p_{12}p_8}$的垂直平分线与$\overline{p_{12}p_9}$的垂直平分线的交点)，$\overline{q_7q_8}$为$\overline{p_8p_9}$的垂直平分线上的一条线段(q_8 为$\overline{p_8p_4}$的垂直平分线与$\overline{p_8p_4}$的垂直平分线的交点)，q_8 向下的射线是$\overline{p_9p_4}$的垂直平分线上的一条射线。折线 B 的构造过程如图 4-13 所示。

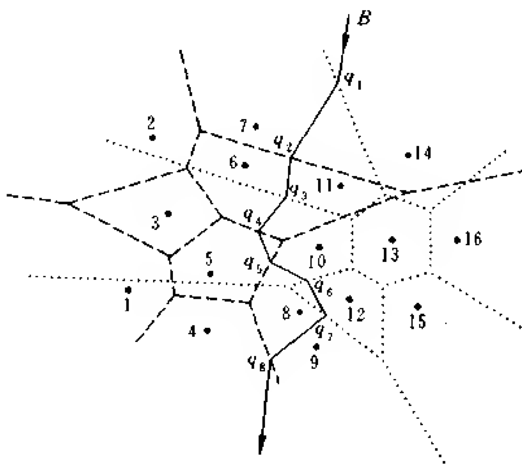


图 4-12 构造 Voronoi 图的分治算法

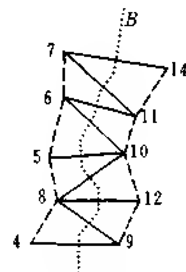


图 4-13 构造折线 B 的过程

总之，该过程可以看成是三角形序列的演变过程，也就是 $p_{14}p_7p_{11} \to p_7p_{11}p_6 \to p_{11}p_6p_{10} \to p_6p_{10}p_5 \to p_5p_{10}p_8 \to p_{10}p_8p_{12} \to p_{12}p_8p_9 \to p_8p_9p_4$，称为三角形顶点转移法。

设 $S_1=\{a_1,a_2,\cdots,a_k\}$，$S_2=\{b_1,b_2,\cdots,b_{k'}\}$，并假设已求得 $CH(S_1)$ 和 $CH(S_2)$。构造折线 B 的算法如下：

步 1 计算 $CH(S_1)$ 和 $CH(S_2)$ 的正切线，设为 $\overline{p_{a_1}p_{b_1}}$ 和 $\overline{p_{a_k}p_{b_{k'}}}$，$p_{a_1}$ 的 y 坐标大于 p_{a_k} 的 y 坐标，p_{b_1} 的 y 坐标大于 $p_{b_{k'}}$ 的 y 坐标。

步 2 作 $\overline{p_{a_1}p_{b_1}}$ 的垂直平分线 $l_{a_1b_1}$，$l_{a_1b_1}$ 与 p_{b_1}（或 p_{a_1}）关联的 Voronoi 多边形边（$\overline{p_{b_1}p_{b_2}}$ 的垂直平分线或 $\overline{p_{a_1}p_{a_2}}$ 的垂直平分线）相交。如果有多个交点，则取 y 坐标值最大的点为 B 的第 1 个顶点 q_1；否则，交点为 B 的新顶点。

步 3 用三角形顶点转移法选择新的三角形，并用步 2 的方法计算 B 的新顶点，直至作出 $\overline{p_{a_k}p_{b_{k'}}}$ 的垂直平分线。

执行步 2 时需要确定 $l_{a_1b_1}$ 与 p_{b_1} 关联的 Voronoi 多边形边的哪条边相交，这只要判断该 Voronoi 多边形的端点对是否位于 $l_{a_1b_1}$ 的两侧，如果位于两侧，则相交；否则，不相交。如果相交，则求出交点，得到 B 的一个新顶点。

步 1 需要线性时间。折线 B 穿过 $Vor(S_1)$ 与 $Vor(S_2)$ 时，组成 B 的线段数目不超过 $|S_1|+|S_2|=n$。求每条线段只需常数时间，所以构造 B 仅用线性时间。设 $T(n)$ 表示用分治法构造 n 个点的点集 S 的 Voronoi 图所需要的时间，则 $T(n)$ 满足下述递归关系式

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

其中 $O(n)$ 为合并 $Vor(S_1)$ 和 $Vor(S_2)$ 所需要的时间。该递归关系式的解为 $T(n)=O(n\log n)$。

4.2.4 减量算法

已知点集 $S=\{p_1,p_2,\cdots,p_n\}$ 的 Voronoi 图，现删去点 p_i 之后，要求构造 Voronoi 图 $Vor(\{p_1,p_2,\cdots,p_{i-1},p_{i+1},\cdots,p_n\})$。

如果 p_{i-1},p_i,p_{i+1} 是 $BCH(S)$ 上的连续顶点，则删去点 p_i 及 p_i 关联的 Voronoi 多边形的边和顶点。之后，如果 p_{i-1} 与 p_{i+1} 成为 $BCH(S-\{p_i\})$ 上相邻顶点，并设 p_i 关联的 Voronoi 多边形的边为 $e_1,e_2,\cdots,e_k$，这些边分别是 p_i 与 p_j，p_i 与 p_{j+1}，$\cdots$，p_i 与 p_{j+k} 的垂直平分线，那么作 $\overline{p_{i-1}p_{i+1}}$ 的垂直平分线并且修改点 $p_{i-1},p_j,p_{j+1},\cdots,p_{j+k},p_{i+1}$ 关联的 Voronoi 多边形的边和顶点，便可求得 $Vor(\{p_1,p_2,\cdots,p_{i-1},p_{i+1},\cdots,p_n\})$。图 4-14(a)中删去点 p_5 及 p_5 关联的 Voronoi 多边形的边和顶点，之后，点 p_4 与 p_1 成为 $BCH(\{p_1,p_2,p_3,p_4\})$ 上相邻的顶点，点 p_5 关联的 Voronoi 多边形的边分别是 p_5 与 p_1，p_5 与 p_2，p_5 与 p_4 的垂直平分线，作 $\overline{p_1p_4}$ 的垂直平分线，并且修改点 p_4，p_1，p_2 关联的 Voronoi 多边形的边和顶点，最后以点线表示 $Vor(\{p_1,p_2,p_3,p_4\})$。

如果删去点 p_i 之后，p_{i-1} 与 p_{i+1} 不是 $BCH(S-\{p_i\})$ 上相邻顶点，如图 4-11(b)所示，那么只要删去点 p_i（图 4-11(b)中点 p）及 p_i 关联的 Voronoi 多边形的边和顶点，并修改 $BCH(S-\{p_i\})$ 上新顶点关联的 Voronoi 多边形的边和顶点，便可得到 $Vor(\{p_1,p_2,\cdots,p_{i-1},p_{i+1},\cdots,p_n\})$。图 4-11(b)中，删去点 p 之后，点 p_1 是 $BCH(S-\{p\})$ 上新顶点，点线

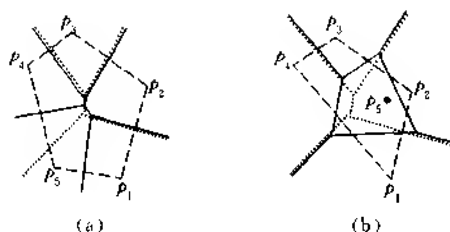


图 4-14　构造 Voronoi 图的减量算法

为删去的点 p 关联的 Voronoi 多边形边，虚线为 Vor($\{p_1,p_2,p_3,p_4\}$)。

考虑删去的点 p_i 在凸壳内部，如图 4-14(b)中的点 p_5，此时删去点 p_5 关联的 Voronoi 多边形边和顶点，并修改点 p_1、p_2、p_3、p_4 关联的 Voronoi 多边形边和顶点，因为这些 Voronoi 多边形与 p_5 关联的 Voronoi 多边形有共同的边。

构造点集 S 的 Voronoi 图的减量算法

步 1　**if**　p_{i-1},p_i,p_{i+1}是 BCH(S)上连续的 3 个顶点 ∧ p_{i-1}与 p_{i+1}是 BCH($S-\{p_i\}$)上相邻顶点 ∧ p_i 关联的 Voronoi 多边形边 $e_{i_1},e_{i_2},\cdots,e_{i_k}$分别是 p_i 与 p_j，p_i 与 p_{j+1}，…，p_i 与 p_{j+k}的垂直平分线。

then　删去点 p_i 及 p_i 关联的 Voronoi 多边形边和顶点，作$\overline{p_{i-1}p_{i+1}}$的垂直平分线并修改点 $p_{i-1},p_j,p_{j+1},\cdots,p_{j+k},p_{i+1}$关联的 Voronoi 多边形的边和顶点。

else if　p_{i-1}、$p_{k'}$、p_{i+1}是 BCH($S-\{p_i\}$)上连续的 3 个顶点。

then　删去点 p_i 及 p_i 关联的 Voronoi 多边形的边和顶点，并修改点 $p_{k'}$ 关联的 Voronoi 多边形的边和顶点。

步 2　**if**　p_i 在 CH(S)内部 ∧ p_i 关联的 Voronoi 多边形边分别是 p_i 与 p_j，p_i 与 p_{j+1}，…，p_i 与 p_{j+k}的垂直平分线。

then　删去点 p_i 及 p_i 关联的 Voronoi 多边形边和顶点，并修改点 $p_j,p_{j+1},\cdots p_{j+k}$关联的 Voronoi 多边形的边和顶点。

执行该算法时，首先判定点 p_i 是否为凸壳顶点或在凸壳内，这只要求出 CH(S)，再进行比较，其耗费为 $O(n\log n)$。如果点 p_{i-1}、p_i、p_{i+1}是 BCH(S)上连续的 3 个点，删去点 p_i 之后，耗费 $O(\log^2 n)$时间可以恢复 BCH($S-\{p_i\}$)，即判定 p_{i-1}与 p_{i+1}之间是否有新的凸壳顶点。删去点 p_i 及 p_i 关联的 Voronoi 多边形的边和顶点，修改相应的 Voronoi 多边形的边和顶点，耗费常数时间。因此算法的时间复杂性为 $O(n\log n)$。

4.2.5　平面扫描算法

构造 Voronoi 图的平面扫描算法是 Fortune(1987)提出的，其复杂性是 $O(n\log n)$。

平面扫描算法通过平面上的一条扫描线由左向右扫描，在平面上已扫描过的部分就得到问题的解，而未扫描部分还没有形成问题的解。要使构造 Voronoi 图的平面扫描算法能在线已扫过的部分构造出 Voronoi 图，其困难是扫描线 L 在碰到决定 Voronoi 域$V(p_i)$的点 p_i 之前会遇到该域的 Voronoi 边。为克服这一困难，先介绍下面的概念。

设点 p 位于三维坐标系的 xy 平面上，顶点在 p 并且其侧面以 45°倾斜的圆锥体垂直于 xy 平面。如果将第三个变元看成是时间，那么以 p 为顶点的圆锥体表示以单位速度在 p 周围扩张的圆，t 个单位时间之后，圆的半径为 t。

考虑以点 p_1 和 p_2 为顶点的两个圆锥体 $Con(p_1)$和 $Con(p_2)$，它们在三维空间中相交成一条曲线，该曲线位于垂直于 xy 平面的平面上，这个平面与 xy 平面的交是$\overline{p_1p_2}$的垂直平分线。因此，虽然两个圆锥体的交线在三维空间中是一条曲线，但该曲线投影到 xy 平面上却是一条直线，而且是 $\mathrm{Vor}(\{p_1,p_2\})$。同样，以点 p_1，p_2 和 p_3 为顶点的三个圆锥体的交(3 条曲线)在 xy 平面上的投影形成 $\mathrm{Vor}(\{p_1,p_2,p_3\})$。

构造 Voronoi 图的平面扫描算法的基本想法是，通过与 xy 平面成 45°倾斜的平面 π 扫描锥体，π 与 xy 平面的交线作为扫描线 L。假设 L 平行于 y 轴，并且它的 x 坐标是 l，如图 4-15 所示；另设平面 π 和锥体是不透明的，并从 $z=-\infty$向上观察平面 π 和锥体。

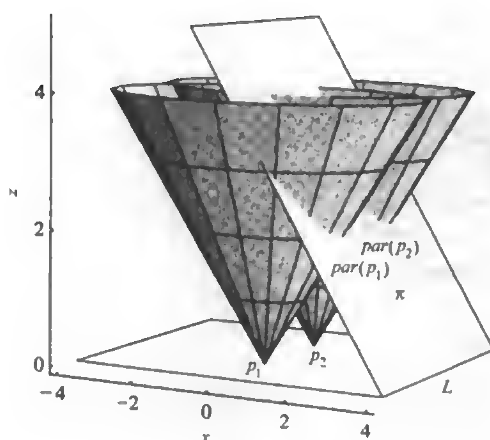


图 4-15 扫描平面 π 切割锥体，π 和 L 向右扫描

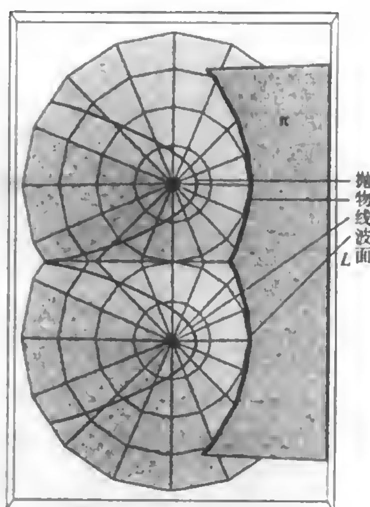
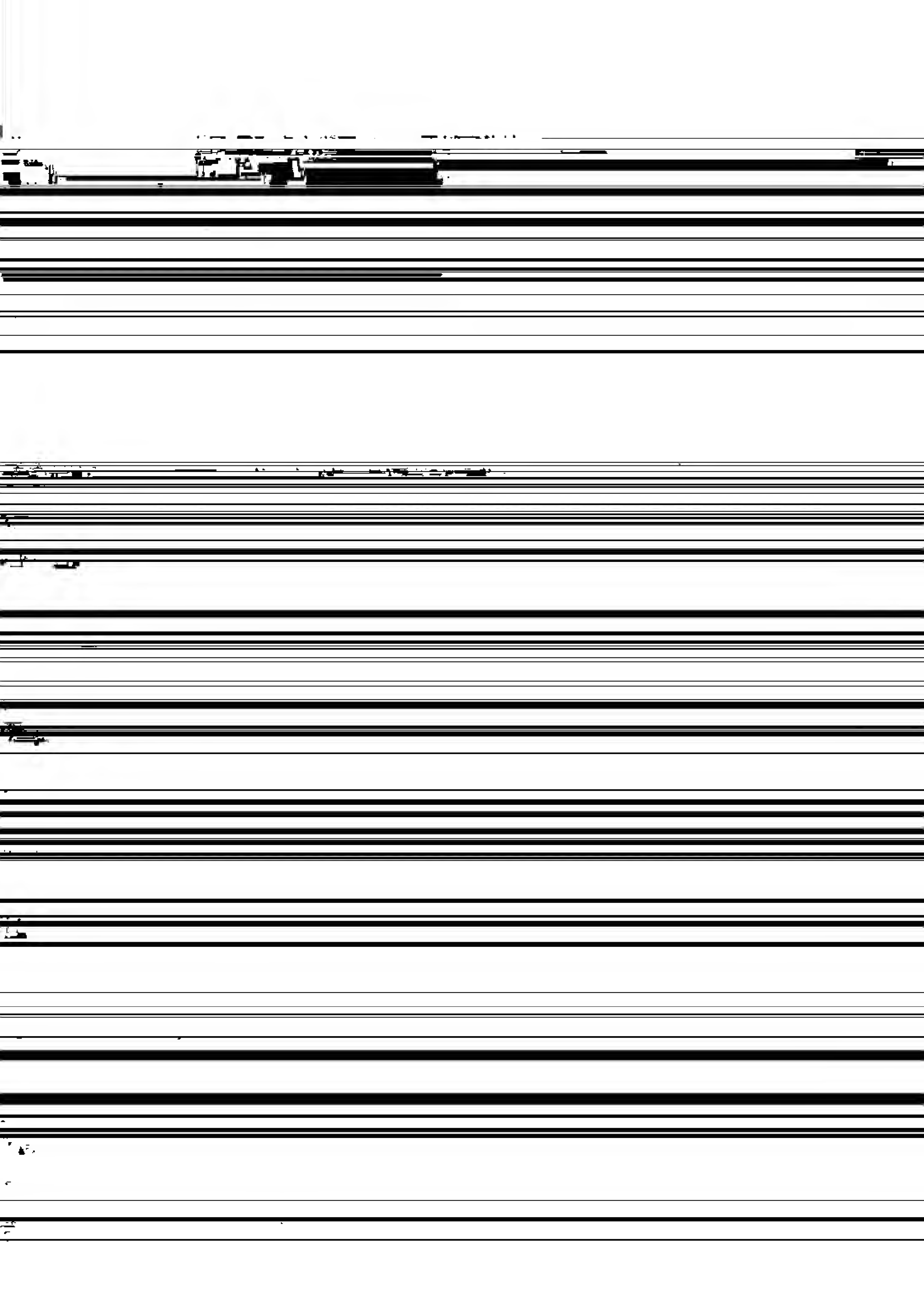


图 4-16 随着 π、L 的右移，抛物线波面的变迁

图 4-15 中，L 的 $x=0$(点 p_1、p_2 的 x 坐标为 0)，此时从下往上观察，只能看见平面 π，点 p_1 和 p_2 在扫描线 L 上，但看不见锥体，这时是扫描的起始时刻。

随后，扫描平面 π 和扫描线 L 向右平移，在平移过程中，平面 π 切割 $Con(p_1)$、$Con(p_2)$分别形成抛物线 $Par(p_1)$和 $Par(p_2)$，然后将其投影到 xy 平面，称为抛物线波面，如图 4-16 所示。随着 π、L 的右移，两条抛物线 $Par(p_1)$，$Par(p_2)$的交的轨迹构成一条抛物线 $Par(Con(p_1)\cap Con(p_2))$，该抛物线在 xy 平面上的投影即$\overline{p_1p_2}$的垂直平分线。从 $z=-\infty$向上观察，抛物线 $Par(Con(p_1)\cap Con(p_2))$是一条直线段，并且是 $\mathrm{Vor}(\{p_1,p_2\})$。在平面 π 和线 L 已扫描过的部分，点 p_1 和 p_2 及部分 $Par(Con(p_1)\cap Con(p_2))$均已形成，而未扫描部分，$Par(Con(p_1)\cap Con(p_2))$的剩余部分还未形成。直至平面 π 离开 $Con(p_2)$时，扫描终止，$\mathrm{Vor}(\{p_1,p_2\})$完全形成。



…,8。S 的凸壳顶点即 S 中全部点，三角剖分 CH(S)为 6 个三角形，因此从 Voronoi 点 v_1 向下的折线由 5 条线段组成：$\overline{v_1v_2}$，$\overline{v_2v_3}$，$\overline{v_3v_4}$，$\overline{v_4v_5}$，$\overline{v_5v_6}$。该 Voronoi 图有 13 条 Voronoi 边：$l_1, l_2, \cdots, l_8$ 及 5 条折线段；6 个 Voronoi 点：$v_1, v_2, \cdots, v_6$。

$Z_{4\text{-}1}$ 算法中步 1 需要时间 $O(n\log n)$，步 2 耗费 $O(n)$。步 3 中每次耗费常数时间可以求得一条对角线，共有 $O(n)$ 条对角线，因此，每次用最长对角线三角剖分凸多边形需要 $O(n)$ 时间。步 4 耗费常数时间，步 5 用线性时间可以求得折线，步 7 与步 8 耗费线性时间。因此步 1 至步 8 的时间复杂性为

$$O(n\log n) + O(n) + O(n) + O(n) + O(n) = O(n\log n)$$

一般情况下，如果每次都用最长对角线分割凸多边形，那么步 9 获得肯定答案的可能性是大的。因此，需要重新三角剖分凸多边形的可能性较小。

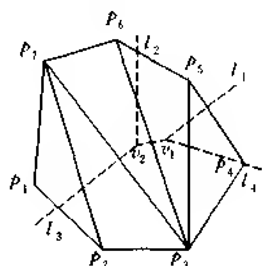


图 4-18　最远点意义下的 Voronoi 图(放宽条件下)

如果允许最远点意义下 Voronoi 点的数目小于凸壳三角剖分的三角形数目，或Voronoi多边形数目小于凸壳顶点数目，那么步 9 获得肯定答案的可能性将增加。如图 4-18 所示，图中只有两个 Voronoi 点(三角形数目为 5)，4 个 Voronoi 多边形(凸壳顶点数是 7)。

4.3　平面点集的三角剖分

平面上给定 n 个点 $p_1, p_2, \cdots, p_n$，所谓平面点集三角剖分是指用互不相交的直线段连接 p_i 与 p_j，$1 \leqslant i, j \leqslant n, i \neq j$，并使凸壳内的每一个区域是一个三角形，如图 4-19 所示。

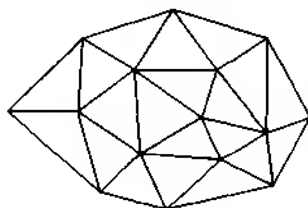


图 4-19　平面点集的一种三角剖分

由于三角剖分是一个平面图，它有 n 个顶点，因此该图至多有 $3n-6$ 条边。如果能给出这些边的一个表，那么就得到问题的一个解。对三角剖分问题可以提出许多约束条件，例如，最小角最大化；边的总长度最小化(称为最小权三角剖分)等。在许多应用中，最好是三角形尽可能为"等边的"，或边的总长度最小。

由 4.1 节和 4.2 节可以知道，无论是最近点意义下的 Voronoi 图，还是最远点意义下的 Voronoi 图都与点集的三角剖分有着密切的关系：最近点意义下 Voronoi 图的对偶图就是点集的一种三角剖分，因而由点集的三角剖分可以计算 Voronoi 图；最远点意义下 Voronoi 图的对偶图是点集凸壳(凸多边形)的一种三角剖分，由点集凸壳的三角剖分可以求得 Voronoi 图(只要该 Voronoi 图存在)。当然，点集的三角剖分不仅仅是构造 Voronoi 图的准备工作，而且它本身有许多应用。本节介绍三角剖分的几种算法。

4.3.1　平面点集三角剖分的贪心算法

本小节介绍的算法均要求三角剖分后边的总长度尽可能小。算法的基本思想是先将所有的两点间距离从小到大排序，依次序每次取一条三角剖分的边，直至达到要求的边

数。下述贪心算法 1 和贪心算法 2 的区别在于不同的终止条件，因而有不同的复杂性。但两个算法中都要判断新加入的边与已加入的边是否相交，只有不相交的边才能成为三角剖分新加入的边。

贪心算法 1

步 1 $T\leftarrow\varnothing$

步 2 计算点集 S 中所有点对之间的距离 $d(p_i,p_j)$，$1\leqslant i,j\leqslant n,i\neq j$，并且对距离进行分类，设为 $d_1,d_2,\cdots,d_{\frac{n(n-1)}{2}}$，相应的线段记为 $e_1,e_2,\cdots,e_{\frac{n(n-1)}{2}}$。

步 3 $k\leftarrow1$。

步 4 **if** e_k 与 T 中的边不相交 **then** $T\leftarrow e_k$ **else** 删去 e_k。

步 5 $k\leftarrow k+1$，**goto** 步 4，直至 $k=\dfrac{n(n-1)}{2}$。

步 6 输出 T。

该算法中，T 存储三角剖分的边。步 2 需要计算 $\frac{n(n-1)}{2}$ 次距离，另外距离分类需要 $O(n^2\log n)$ 次比较。T 中元素随步 5 至步 4 循环次数的增加而增加，因此向 T 中加入一条新边所需要的判定两条线段是否相交的次数也随之增加。如果步 5 至步 4 的前 $3n-6$ 次循环后已构成点集的三角剖分，那么步 5 至步 4 循环所需要的判定两条线段是否相交的次数为

$$1+2+\cdots+3n-7+(3n-6)\times\left(\frac{n(n-1)}{2}-(3n-6)\right)=O(n^3)$$

在常数时间内可以判定两条线段是否相交，因此贪心算法 1 的时间复杂性为 $O(n^3)$。

贪心算法 2

步 1 $T\leftarrow\varnothing$

步 2 计算点集 S 中所有点对之间的距离 $d(p_i,p_j)$，$1\leqslant i,j\leqslant n,i\neq j$，并且分类距离，设为 $d_1,d_2,\cdots,d_{\frac{n(n-1)}{2}}$，相应的线段记为 $e_1,e_2,\cdots,e_{\frac{n(n-1)}{2}}$。

步 3 $k\leftarrow1,l\leftarrow0$。

步 4 **if** e_k 与 T 中的边不相交

then $T\leftarrow e_k,l\leftarrow l+1$

else 删去 e_k。

步 5 $k\leftarrow k+1$，**goto** 步 4，直至 $l>3n-6$。

步 6 输出 T。

执行贪心算法 2 时，如果步 5 至步 4 的前 $3n-6$ 次循环后已构成点集的三角剖分，那么步 5 至步 4 循环所需要的判定两条线段是否相交的次数为

$$1+2+\cdots+(3n-7)=O(n^2)$$

而步 2 的复杂性为 $O(n^2\log n)$，所以贪心算法 2 的复杂性为 $O(n^2\log n)$。

条件“步 5 至步 4 的前 $3n-6$ 次循环后已构成点集的三角剖分”不成立时，贪心算法 2 的复杂性可以达到 $O(n^3)$。

下面再介绍另一种形式的贪心算法。

贪心算法 3

步 1 求点集凸壳，设凸壳顶点为 $p_1,p_2,\cdots,p_m$，凸壳的边为 $e_1,e_2,\cdots,e_m$。$e_i(i=\overline{1,m})$ 加入 T（三角剖分的边集合），并且 e_i 的权值被赋为 1。凸壳内点的集合为 $S_1=\{p_{m+1},p_{m+2},\cdots,p_n\}$。

步 2 从 S_1 中任取一点 p_i，求与 p_i 距离最近的点，设为 p_j，$\overline{p_ip_j}$加入 T。

步 3 求与 p_j 距离最近的点（除点 p_i 外），设为 p_k，$\overline{p_jp_k}$加入 T，$\overline{p_kp_i}$加入 T，$p_ip_jp_k$ 构成一个三角形，其边的权值 w_{ij},w_{jk},w_{ki}均被赋值 1。

步 4 分别求与 p_i,p_j,p_k 距离最近的点（除点 p_i,p_j,p_k 外），设为 p_i',p_j',p_k'，$\overline{p_ip_i'}$、$\overline{p_i'p_k}$、$\overline{p_jp_j'}$、$\overline{p_j'p_i}$、$\overline{p_kp_k'}$与$\overline{p_k'p_j}$加入 T，并且这些边的权值被赋值 1，而 w_{ij},w_{jk},w_{ki}的值加 1，即为 2（权值为 2 的边，即为两个三角形所共有）。

步 5 对权值为 1 的边（除 $e_1,e_2,\cdots,e_m$ 外）的两个端点分别求与其距离最近的点，并将其连线（得到新的三角形）加入 T，新三角形边的权值加 1。

步 6 对权值为 1 的边重复步 5，直至所有边的权值为 2（除 $e_1,e_2,\cdots,e_m$ 外）。

图 4-20 是贪心算法 3 的一个执行例子。

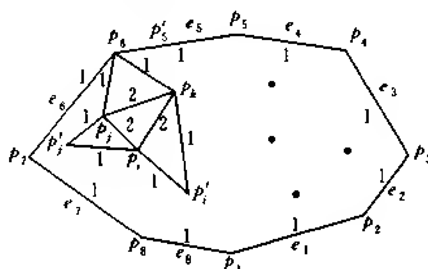


图 4-20　贪心算法 3 的示意图

贪心算法 3 中，步 1 耗费 $O(n\log n)$，步 2 需要计算 $n-1$ 次距离和 $n-2$ 次比较。步 3 求 p_k 要计算 $n-2$ 次距离和 $n-3$ 次比较，步 4 要进行$(n-3)\times3$ 次距离计算及$(n-4)\times3$ 次比较。步 5 至多进行$(n-6)$次距离计算与$(n-7)$次比较。步 6 至步 5 的循环次数不超过 $3n-9$，因此贪心算法 3 的时间复杂性为

$$O(n\log n)+O(n)+O(n)+O(n)+(n-6)\times(3n-9)=O(n^2)$$

4.3.2 Delaunay 三角剖分与多边形内部点集的三角剖分

最近点意义下的 Voronoi 图的对偶图实际上是点集的一种三角剖分，该三角剖分就是 Delaunay 三角剖分（表示为 $DT(S)$），其中每个三角形的外接圆不包含点集中的其他任何点。因此，在构造点集的 Voronoi 图之后，再作其对偶图，即对每条 Voronoi 边（限有限长线段）作通过点集中某两点的垂线，便得到 Delaunay 三角剖分。也可以用类似于构造 Voronoi 图的递归过程直接构造 Delaunay 三角剖分，而且时间复杂性为 $O(n\log n)$。

当多边形内部不包含任何点时，可以利用 2.3 节中的方法进行三角剖分或先利用2.4 节中的方法将多边形划分成凸多边形，然后再把凸多边形划分成三角形。

如果多边形内部包含其他点，那么可用下面的算法进行三角剖分，该算法是周培德于

1995 年提出的。

$Z_{4\text{-}2}$算法

输入 任意简单多边形 L 的顶点序列 $p_1, p_2, \cdots, p_n$ 与 L 内的点集 $S=\{q_1, q_2, \cdots, q_m\}$。

输出 L 的内域划分成若干个三角形，其顶点取自$\{p_1, p_2, \cdots, p_n\}$和$\{q_1, q_2, \cdots, q_m\}$，三角形内及边界上不含$\{p_1, p_2, \cdots, p_n\}$和 S 的中点。

步 1 求$\{p_1, p_2, \cdots, p_n\}$的凸壳，设凸壳 $C_1=\{r_1^1, r_2^1, \cdots, r_{a_1}^1\}$，$(a_1 \leqslant n)$。

步 2 **if** $a_1=n$ **then** 按算法 $Z_{4\text{-}3}$划分 L 的内域成三角形序列，并输出结果，终止。

else 从 L 中找出所有不同于$\{r_1^1, r_2^1, \cdots, r_{a_1}^1\}$的点 p_j 及点列 $p_l, p_{l+1}, \cdots, p_{l+k}$。$p_j$ 为凹点，p_{j-1}, p_{j+1}为凸点，而 $p_l, p_{l+1}, \cdots, p_{l+k}$中必有凹点。转步 3。

步 3 $D_1 \leftarrow \{q_1, q_2, \cdots, q_m\} \cup \{p_1, p_2, \cdots, p_n\} - C_1$

步 4 $i \leftarrow 1$

步 5 求 D_i 的凸壳，设凸壳 $C_{i+1}=\{r_1^{i+1}, r_2^{i+1}, \cdots, r_{a_{i+1}}^{i+1}\}$

步 6 **if** C_{i+1}内不含 p_j 及 $p_l, p_{l+1}, \cdots, p_{l+k}$

then 用算法 $Z_{4\text{-}3}$划分 C_{i+1}的内域及 C_i 与 C_{i+1}之间的环域成三角形序列。$d \leftarrow i+1$，转步 12。

else 转步 7。

步 7 **if** p_j(或 $p_l, p_{l+1}, \cdots, p_{l+k}$)在 C_{i+1}的内部 $\wedge \overline{q_{i'}q_{j'}}$($C_{i+1}$的边)与$\overline{p_{j-1}p_j}$，$\overline{p_jp_{j+1}}$相交。

then 删去$\overline{q_{i'}q_{j'}}$，连接 p_{j-1}与 $q_{i'}$，p_j与 $q_{i'}$，$q_{j'}$与 p_{j+1}，$q_{j'}$与 p_j(如果三角形 $p_jp_{j-1}q_{i'}$内有 S 中点 q_i，则 q_i 与三个顶点 $p_j, p_{j-1}, q_{i'}$连接；如有多个点，则递归地进行连接)；类似处理 $p_l, p_{l+1}, \cdots, p_{l+k}$。转步 8。

else if p_j 为 C_{i+1}的顶点(或 $p_l, p_{l+1}, \cdots, p_{l+k}$为 C_{i+1}的顶点)

then 不进行任何操作，转步 8。

步 8 $D_{i+1} \leftarrow D_i - C_{i+1}$，$i \leftarrow i+1$，步 7 中用$\overline{p_jq_{i'}}$代替$\overline{p_{j-1}p_j}$，$\overline{p_jq_{j'}}$代替$\overline{p_jp_{j+1}}$，转步 5，直至$|D_d|=0,1,2$(即 C_d 内含 0,1,2 个 L 的顶点)，转步 9～步 11。

步 9 用算法 $Z_{4\text{-}3}$分割 C_d 的内域及 C_i 与 C_{i+1}之间的环域(如果 C_d 的顶点全部是 L 的顶点，则不分割 C_d 的内域；如果 C_i 与 C_{i+1}的顶点全部(或部分)是 L 的顶点，则不分割 C_i 与 C_{i+1}之间的环域(或部分环域))，$d \leftarrow i+1$，转步 12。

步 10 **if** C_d 内含点 p_d **then** p_b 与 C_d 的各顶点连接，用算法 $Z_{4\text{-}3}$分割 C_i 与 C_{i+1}之间的环域，$d \leftarrow i+1$，转步 12。

步 11 设 p_1, p_2 是 C_d 内的两个点，p_1(或 p_2)与 C_d 的顶点连接(连接与 p_1(或 p_2)关联的 L 边不相交)，再连接 p_2(或 p_1)与 C_d 的一个顶点(p_1, p_2 与 C_d 的两个顶点构成四边形)。用算法 $Z_{4\text{-}3}$分割各环域，$d \leftarrow i+1$，转步 12。

步 12 $i \leftarrow d$

步 13 查 C_i 的边$\overline{r_j^ir_{j+1}^i}$，以$\overline{r_j^ir_{j+1}^i}$为公共边的两个三角形的顶点设为 $r_i^{i+1}, r_{i'}^{i-1}$。

if $|\overline{r_i^{i+1}r_{i'}^{i-1}}| < |\overline{r_j^ir_{j+1}^i}|$ **then** 连接 r_i^{i+1} 与 $r_{i'}^{i-1}$，删去$\overline{r_j^ir_{j+1}^i}$。

else　不改变连接方式。

步 14　$i \leftarrow i-1$，转步 13，直至 $i=1$，输出结果，终止。

$Z_{4\cdot2}$算法的步 1 是求 L 顶点的凸壳 C_1，C_1 的顶点不可能是 S 中的点。当 L 是凸多边形时，即 $a_1=n$，用算法 $Z_{4\cdot3}$可以把 L 内点集 S 三角剖分，而且可以得到最小权（或次最小权）三角剖分。如果 L 不是凸多边形，即 $a_1<n$，则要找出位于 C_1 内部的 L 顶点子序列 p_l，p_{l+1}，…，p_{l+k}。p_j 及 p_l，p_{l+1}，…，p_{l+k}必位于互相嵌套的多层凸壳 $C_i(i=\overline{1,d})$的内部，此时凸壳 C_i 的某些边（比如$\overline{q_iq_j}$）必与 p_j 关联的边（$\overline{p_{j-1}p_j}$，$\overline{p_jp_{j+1}}$）相交，步 7 和步 8 是专为处理这种情况而设计的。算法的步 9～步 11 分别处理最内层凸壳 C_d 内不含或含 1 个、2 个 L 的顶点时的情况。步 12～步 14 使三角剖分最小化（即改变连接方式，使所有三角形的边长之和最小）。因此该算法正确地将 L 内点集 S 三角剖分，而且能得到最小权（或次最小权）三角剖分。

$Z_{4\cdot2}$算法的步 7，步 10 与步 11 的工作是解决 C_i 内包含点 p_j 及子序列 p_l，p_{l+1}，…，p_{l+k}的情况，此时耗费线性次比较。除此之外，该算法的耗费与算法 $Z_{4\cdot3}$的耗费相同。

$Z_{4\cdot2}$算法用于凸多边形内点集、任意多边形内点集以及带有内孔的任意多边形内点集等多种不同情况，均可获得最短长度（或次最短长度）的三角剖分，如图 4-21 所示。

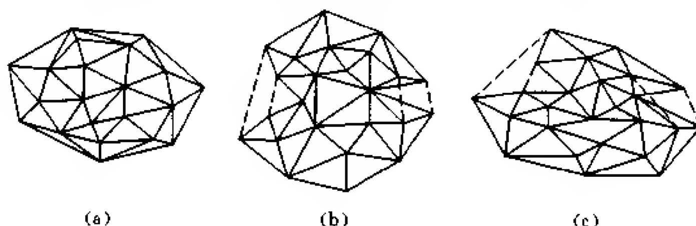


图 4-21　$Z_{4\cdot2}$算法应用举例

4.3.3　平面点集三角剖分的算法

给定平面上 n 个点，用不相交的直线段连接它们，使该 n 个点的凸壳内的每一个区域是一个三角形，这就是三角剖分问题。当三角剖分边的总长度减至最小时，则称为最小权三角剖分。无论是 Delaunay 三角剖分、贪心法三角剖分、依据环形试探法设计的三角剖分算法，还是最近 Eppstein 提出的一种三角剖分算法均不能产生最小权三角剖分。下面介绍的算法是周培德于 1996 年提出的。

$Z_{4\cdot3}$算法（平面点集三角剖分的基本算法）

输入　平面上 n 个点的坐标(x_i,y_i)，$i=\overline{1,n}$，该点集记为 S。

输出　n 个点的三角剖分链表 $L=(p_1,p_2,p_i)\rightarrow(p_2,p_i,p_j)\rightarrow\cdots\rightarrow(p_{n-2},p_{n-1},p_n)$。

步 1　求 n 个点的凸壳顶点，设为 $C_1=\{p_1^1,p_2^1,\cdots,p_{m_1}^1\}$。

if　$m_1=n$　then　三角剖分凸多边形，输出结果。终止。

else　求$\{S-C_1\}$的凸壳顶点，设为 $C_2=\{p_1^2,p_2^2,\cdots,p_{m_2}^2\}$。继续下去，直至求得 $C_m=\{p_1^m,p_2^m,\cdots,p_{m_m}^m\}$。$C_m$ 内不含 S 中的点、包含 S 中的 1 个点或 2 个点，分别转步 2、步 6 和步 8。

步 2 求 C_m 的直径，设 $l(p_i^m, p_j^m)$ 是 C_m 的直径。

步 3 **if** C_m 中有 m_m-1 个点共线（p_1^m 不在线上）。

then 连接 p_1^m 与线上各点，转步 9。

else if $(|\overline{p_{i-1}^m p_{i+1}^m}| < |\overline{p_i^m p_{i+2}^m}|) \wedge (|\overline{p_{i-1}^m p_{i+1}^m}| < |\overline{p_i^m p_{i-2}^m}|)$

then 连接 p_{i-1}^m 与 p_{i+1}^m，删去点 p_i^m，输出 $(p_i^m, p_{i+1}^m, p_{i-1}^m)$。

else 连接 p_i^m 与 p_{i+2}^m（或 p_i^m 与 p_{i-2}^m），删去点 p_{i+1}^m（或点 p_{i-1}^m），输出 $(p_{i-1}^m, p_{i+2}^m, p_i^m)$ 或输出 $(p_{i-1}^m, p_{i-2}^m, p_i^m)$。

if $(|\overline{p_{j-1}^m p_{j+1}^m}| < |\overline{p_j^m p_{j+2}^m}|) \wedge (|\overline{p_{j-1}^m p_{j+1}^m}| < |\overline{p_j^m p_{j-2}^m}|)$

then 连接 p_{j-1}^m 与 p_{j+1}^m，删去点 p_j^m，输出 $(p_j^m, p_{j+1}^m, p_{j-1}^m)$。

else 连接 p_j^m 与 p_{j+2}^m（或 p_j^m 与 p_{j-2}^m），删去点 p_{j+1}^m（或点 p_{j-1}^m），输出 $(p_{j-1}^m, p_{j+2}^m, p_j^m)$ 或输出 $(p_{j-1}^m, p_{j-2}^m, p_j^m)$。

步 4 $C_m - \{p_i^m, p_j^m\}$（或 $C_m - \{p_{i+1}^m, p_{j+1}^m\}, C_m - \{p_{i-1}^m, p_{j-1}^m\}$），记为 C_m^1。

步 5 求 C_m^1 的直径，设 $l(p_{i'}^m, p_{j'}^m)$ 是 C_m^1 的直径，分别以 $p_{i'}^m$、$p_{j'}^m$ 代替 p_i^m、p_j^m，重复步 3、步 4、步 5，直至分割完毕。转步 9。

步 6 设 C_m 内含 1 个点 p。C_m 各顶点与 p 连接，并按连线长度排序 $d_1^m, d_2^m, \cdots, d_{m_m}^m$，其中 $d_1^m(p_1^m, p)$ 最大。

步 7 依次以 $p_1^m, p_2^m, \cdots$ 为顶点，并当 $|\overline{p_i^m p}| > |\overline{p_{i-1}^m p_{i+1}^m}|$ $(i=1,2,\cdots)$ 时，用步 3 至步 5（步 5 改为求 C_m 的剩余顶点与 p 的最大距离）的方法（此时不考虑 j）进行分割，直至 $|\overline{p_i^m p}| < |\overline{p_{i-1}^m p_{i+1}^m}| \wedge |\overline{p_i^m p}| < |\overline{p_i^m p_{i+2}^m}| \wedge |\overline{p_i^m p}| < |\overline{p_{i-2}^m p_i^m}|$ 为止。转步 9。

步 8 设 C_m 内含两个点 p_1, p_2，连接 p_1 与 p_2。C_m 中位于 $\overrightarrow{p_1 p_2}$ 右侧的点，设为 $p_1^m, p_2^m, \cdots, p_i^m$，连接 p_1 与 p_i^m，p_2 与 p_1^m（$\overline{p_1 p_i^m}$ 与 $\overline{p_2 p_1^m}$ 不相交）。$p_1 p_2 p_1^m \cdots p_i^m$ 成一凸壳，用步 2 至步 5 的方法分割该凸壳。

if p_i^m 在 $\overline{p_1 p_2}$ 的延长线上

then 连接 p_1 与 p_i^m（或 p_2 与 p_i^m），同样处理 C_m 中位于 $\overrightarrow{p_1 p_2}$ 左侧的点。转步 9。

步 9 $i \leftarrow m$

步 10 分割 C_i 与 C_{i-1} 之间的环域。求 C_{i-1} 中位于边 $\overrightarrow{p_j^i p_{j+1}^i}$ 右侧的点链（逆时针方向），分别连接边 $\overline{p_j^i p_{j+1}^i}$ 与链的两个端点，构成凸壳，用步 2 至步 5 的方法分割该凸壳。同样方法处理环域的其他区域。

步 11 $i \leftarrow i-1$，转步 10，直至 $i=1$。转步 13。

步 12 设 $p_1 p_2 p_3$ 与 $p_3 p_2 p_4$ 是两个有一条公共边 $\overline{p_2 p_3}$ 的三角形。

if $|\overline{p_2 p_3}| \leqslant |\overline{p_1 p_4}| \vee \overline{p_2 p_3}$ 与 $\overline{p_1 p_4}$ 交于两个三角形的外部

then 不改变原来的三角剖分

else if $|\overline{p_2 p_3}| > |\overline{p_1 p_4}| \wedge \overline{p_2 p_3}$ 与 $\overline{p_1 p_4}$ 交于两个三角形的内部

then 连接 p_1 与 p_4，删去线段 $\overline{p_2 p_3}$。

步 13 对以 C_m 中的边（或已变动的边）为公共边的三角形对，用步 12 的方法检查是否需要改变原有的三角剖分。然后，沿 $C_{m-1}, \cdots, C_2$ 的各条边（或已变动的边）寻找两个有

公共边的三角形对，并用步 12 的方法检查是否需要改变原来的三角剖分，直至所有凸壳的边检查完。终止。

$Z_{4\text{-}3}$算法中某些步骤可以进一步细化或修改。下面证明 $Z_{4\text{-}3}$算法的正确性并分析该算法的时间复杂性。

引理 4-1 设 p 是三角形 ABC 内(或边$\overline{BC}$上)的一点，则三角形 ABC 内存在唯一的三角剖分(亦是最小权三角剖分)，即连接 A 与 p、B 与 p 及 C 与 p(或连接 A 与 p)。

引理 4-2 设 $ABCD$ 是任意四边形，连接距离较小的顶点对便得到四边形 $ABCD$ 内的最小权三角剖分。

引理 4-3 任意凸多边形的直径的端点(例如图 4-22 中点 p_0)只能形成三个权值较小的三角剖分(图 4-22 中虚线表示)。

证明 由于是凸多边形，并且直径$\overline{p_0p_4}$(见图 4-22)的长度最长，所以长度$|\overline{p_0p_1}|$、$|\overline{p_0p_2}|$、$|\overline{p_0p_3}|$、$|\overline{p_0p_4}|$是递增序列，而$|\overline{p_0p_4}|$、$|\overline{p_0p_5}|$、$|\overline{p_0p_6}|$、$|\overline{p_0p_7}|$是递减序列。在该两个序列中，$|\overline{p_0p_2}|$与$|\overline{p_0p_6}|$分别是长度最短的连线。因此只要比较$|\overline{p_1p_7}|$、$|\overline{p_0p_2}|$与$|\overline{p_0p_6}|$的长度，找出最小者，便可得到含点 p_0 的最小权三角剖分。 证毕。

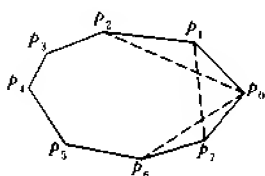


图 4-22 凸多边形

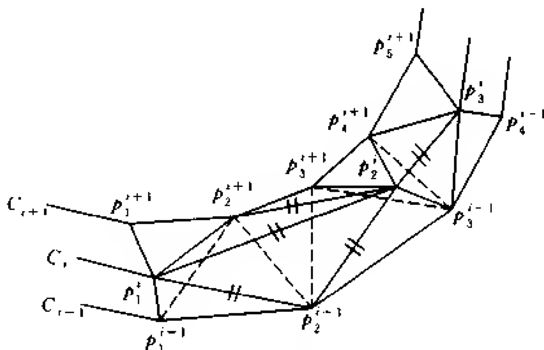


图 4-23 相邻环域的三角剖分

引理 4-4 对 C_i 的边$\overline{p_1^ip_2^i}$执行步 13 至多$(a+1)$次即可求得 C_{i+1}与 C_i、C_i 与 C_{i-1}之间环域中$\overline{p_1^ip_2^i}$段域的最小权三角剖分，如图 4-23 所示。其中 a 为$\overline{p_1^ip_2^i}$段域中 C_{i+1}、C_{i-1}顶点数之和。

证明 对图 4-23 所示的边$\overline{p_1^ip_2^i}$首次执行步 13，即连接 p_2^{i-1} 与 p_2^{i+1}，删去$\overline{p_1^ip_2^i}$；第 2 次执行步 13，即连接 p_2^{i-1} 与 p_3^{i+1}，删去$\overline{p_2^{i+1}p_2^i}$；第 3 次执行步 13，即连接 p_2^{i+1} 与 p_1^{i-1}，删去$\overline{p_1^ip_2^{i-1}}$；第 4 次执行步 13，即连接 p_3^{i+1} 与 p_3^{i-1}，删去$\overline{p_1^{i-1}p_2^i}$。此时 $a=3$，执行 4 次步 13 便将$\overline{p_1^ip_2^i}$段域(C_{i+1}中 p_1^{i+1} 至 p_5^{i+1}、C_{i-1}中 p_1^{i-1} 至 p_3^{i-1} 的边围成的域)进行了最小权三角剖分。对 C_i 的其他边所对应的段域类似处理，就可完成相邻环域的最小权三角剖分。

定理 4-10 给定 n 个点的平面点集 S，$Z_{4\text{-}3}$算法正确地将 S 三角剖分，而且可以得到最小权(或次最小权)三角剖分。$Z_{4\text{-}3}$算法所需要的乘法次数是 $O\left(\frac{n^4}{m^3}\right)$，比较次数是 $O\left(\frac{n^4}{m^3}\right)$，其中 m 为点集凸壳的层数。

证明 显然，算法正确地将 S 三角剖分。

算法的步 1 是分割点集凸壳内的区域成嵌套的环形域，其中 C_m 是最内层的区域。C_m 内可能不含 S 中的点、包含 S 中的 1 个点或 2 个点，但不可能包含 S 中的 3 个点。因为如果包含 S 中的 3 个点，则算法可以再求一次凸壳，得到 C_{m+1}。算法的步 2 至步 5，步 6 至步 7 与步 8 分别处理上述三种情况。其基本方法都是用最短连线从 C_m 中分割出三角形（与贪心思想类似，但不完全相同）。由引理 4-1、引理 4-2 及引理 4-3，C_m 内的三角剖分是最小权（或次最小权）三角剖分。各环域的三角剖分亦同样进行（步 9 至步 11），因此诸环形域的三角剖分是最小权三角剖分。步 13 是为解决 C_m 内的三角剖分及其与相邻环形域三角剖分之间以及相邻环形域三角剖分之间的非最小权问题而设计的。由引理 4-2、引理 4-3 与引理 4-4，算法终止时，可以得到最小权（或次最小权）三角剖分。

$Z_{4\text{-}3}$算法的步 1，第 1 次求凸壳需要 $O(n\log n)$次比较和 $O(n)$次乘法。第 2 次至第 m 次求凸壳所需比较次数和乘法次数均小于 $O(n\log n)$和 $O(n)$。因此步 1 的复杂性不超过 $O(mn\log n)$次比较和 $O(mn)$次乘法。步 2 要进行 $O(m_m)$次求距离（两点间的距离）运算，每次求距离需要 2 次乘法（因为是比较距离的长短，所以可以比较距离的平方值，从而省去开方运算），所以步 2 耗费 $O(m_m)$次乘法和 $O(m_m)$次比较求得 C_m 的直径。设点集 S 有 m 层凸壳，各层凸壳均含 $\left\lfloor \frac{n}{m} \right\rfloor$ 个点，则步 2 需要 $O(n)$次乘法和 $O(n)$次比较。步 3 和步 4 只需要常数时间。第 1 次执行步 5 需要 $O(m_m)$次乘法（即 $O(n)$次乘法）和 $O(m_m)$次比较（即 $O(n)$次比较）便可求得 C_m^1 的直径。如果步 3 至步 5 的每次循环均减少 2 个点，则此循环次数为 $\left\lceil \frac{1}{2}\left(\frac{n}{m}-3\right) \right\rceil$，即 $\left\lceil \frac{n-3m}{2m} \right\rceil$。因此，步 3 至步 5 的耗费不超过 $O(n^2/m)$次乘法和 $O(n^2/m)$次比较。

步 6 耗费$\frac{2n}{m}$次乘法和 $O\left(\frac{n}{m}\log\frac{n}{m}\right)$次比较可以排序$\frac{n}{m}$个元素。步 7 对$\frac{n}{m}$条连线逐条处理，处理一条连线需要求三次点与点的距离，即 6 次乘法，另外再用 2 次比较便可确定一个三角形。因此步 7 需要$\frac{6n}{m}$次乘法及$\frac{2n}{m}$次比较。

步 8 用$\frac{12n}{m}$次乘法可以判定 C_m 中哪些点位于$\overrightarrow{p_1p_2}$的右侧，然后用$\left(\frac{n}{m}-1\right)\frac{2n}{m}+\frac{n-3m}{m}\left(\frac{n}{m}-3\right)\left(\frac{n}{m}-2\right)$次乘法和$\left(\frac{n}{m}-1\right)\frac{n}{m}-1+\frac{n-3m}{m}\left[\frac{1}{2}\left(\frac{n}{m}-3\right)\left(\frac{n}{m}-2\right)-1\right]$次比较完成 C_m 内的三角剖分。

步 9 至步 11 的耗费为步 8 的耗费的$\frac{n}{m}\times m=n$ 倍，即耗费$\frac{12n^2}{m}+\left(\frac{n}{m}-1\right)\frac{n^2}{m}+\frac{n-3m}{2m}\cdot\left(\frac{n}{m}-3\right)\left(\frac{n}{m}-2\right)n$ 次乘法和$\frac{n}{2}\left(\frac{n}{m}-1\right)\frac{n}{m}-1+\frac{(n-3m)n}{2m}\left[\frac{1}{2}\left(\frac{n}{m}-3\right)\left(\frac{n}{m}-2\right)-1\right]$次比较可以完成各环形域的三角剖分。

步 13 是为求得相邻环域最小权三角剖分而设计的，沿 $C_i(i=2,3,\cdots,m)$的各条边（或已变动的边）进行检查。对每条边要求 2 次距离（即 4 次乘法）和一次比较，便可判定以该边为公共边的两个三角形是否需要改变分割方式。由引理 4-4，处理一条公共边执行步 13 至多$(a+1)$次，即求 $2(a+1)$次距离（或 $4(a+1)$次乘法）和$(a+1)$次比较，而 $C_2,C_3,$

…，C_m 的边数之和小于 n，因此重复执行步 13 至多需要 $4(a+1)n<4n^2$ 次乘法和 $(a+1)n<n^2$ 次比较即可完成相邻环域三角剖分的最小权化。

总之，$Z_{4\text{-}3}$算法所需要的乘法次数是：

$$mn + \max\left[O(n) + O(n^2/m), \frac{2n}{m} + \frac{6n}{m}, \frac{12n}{m} + \left(\frac{n}{m} - 1\right)\frac{2n}{m} + \frac{n-3m}{m}\left(\frac{n}{m} - 3\right)\left(\frac{n}{m} - 2\right)\right] + \frac{12n^2}{m}$$

$$+ \left(\frac{n}{m} - 1\right)\frac{n^2}{m} + \frac{n-3m}{2m}\left(\frac{n}{m} - 3\right)\left(\frac{n}{m} - 2\right)n + 4n^2$$

$$\leqslant mn + \max\left[O\left(\frac{n^3}{2m^3}\right), O\left(\frac{n}{m}\right), O\left(\frac{n^3}{m^3}\right)\right] + O\left(\frac{n^4}{m^3}\right) = O\left(\frac{n^4}{m^3}\right)$$

比较次数为：

$$O(m \cdot n\log n) + \max\left[O(n) + O(n^2/m), O\left(\frac{n}{m}\log\frac{n}{m}\right) + \frac{2n}{m}, \frac{n}{m}\left(\frac{n}{m} - 1\right) + \frac{n-3m}{2m}\left(\frac{n}{m} - 3\right)\left(\frac{n}{m} - 2\right)\right] + \frac{n^2}{2m}\left(\frac{n}{m} - 1\right)$$

$$+ \frac{n(n-3m)}{4m}\left(\frac{n}{m} - 3\right)\left(\frac{n}{m} - 2\right) + n^2$$

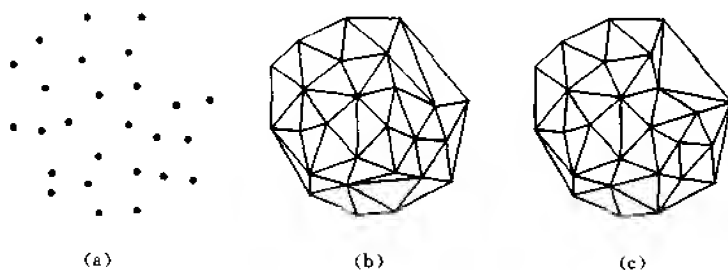
$$\leqslant O(m \cdot n\log n) + \max\left[O\left(\frac{n^3}{m^3}\right), O\left(\frac{n}{m}\log\frac{n}{m}\right), O\left(\frac{n^3}{m^3}\right)\right] + O\left(\frac{n^4}{m^3}\right)$$

$$= O\left(\frac{n^4}{m^3}\right)$$

$m=1$ 时，算法复杂性与算法 CPTA 的复杂性相同。

由于平面上 n 个点的任意三角剖分至多有 $3n-6$ 条边，而 $Z_{4\text{-}3}$算法又可以求得点集的最小权（或次最小权）三角剖分，因此至多只要进行 $3n-7$ 次比较便可求得点集中的最近点对，即求最近点对的复杂性为 $O(n)$。另外，通过 $O((3n-6)\log(3n-6))$ 次比较可以排序点集中某些点对间的距离，对此排序的距离集合，再利用最小生成树算法，可以求得该点集的最小生成树。进而对三角剖分中的每个三角形，作各边的中垂线，即可构造给定点集 S 的 Voronio 图。

将 $Z_{4\text{-}3}$算法应用于图 4-24(a)所示的点集，获得图 4-24(c)所示的剖分结果。



(a) (b) (c)

图 4-24 $Z_{4\text{-}3}$算法应用举例

(a) 点集 S (b) 执行步 1 至 11 的结果 (c) 执行步 13 的结果

4.4 Voronoi 图与三角剖分的应用

本节阐述 Voronoi 图的 6 个应用：最近邻近；最大化最小角三角剖分；最大空圆；最小生成树；货郎担问题；中轴。另外，还介绍 Voronoi 图与凸壳的关系，Voronoi 图的推广，几何数据压缩。

4.4.1 最近邻近

我们可以把最近邻近问题看成是一个查询问题：给定点集 $S=\{p_1,p_2,\cdots,p_n\}$ 及点 q，在 S 中寻找距离 q 最近的点（或者如果 $q\in S$，则在平面上寻找距离 q 最近的点）。另一个问题是所有最近邻近问题：对于给定点集 S 中的每个点 p_i 寻找距离 p_i 最近的邻近点 p_j。这些问题在生物学、生态、地理学和物理学等诸多领域中有许多应用。

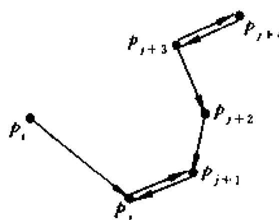


图 4-25 $p_i\to p_j$ 但 $p_j\to p_{j+1}$；另外 $p_{j+3}\to p_{j+4}$，$p_{j+3}\to p_{j+2}$

定义点集 S 中的最近邻近关系如下：p_j 是 p_i 的一个最近邻近，当且仅当 $|p_i-p_j|\leqslant\min\limits_{p\neq p_i}|p_i-p|$，其中 $p\in S$，可以把这个关系写成 $p_i\to p_j$：p_i 的一个最近邻近是 p_j。值得注意的是，就 p_i 和 p_j 所起的作用而论该定义不是对称的，意指该关系本身不是对称的，事实上的确是这种情况：当 $p_i\to p_j$ 成立时，$p_j\to p_i$ 不一定成立，如图 4-25 所示。此外，一个点可能有几个相等的最近邻近，例如，图 4-25 中的点 p_{j+3}。

1. 最近邻近查询

给定一个固定的点集 S，在 $O(n\log n)$ 时间内构造 Voronoi 图。现在对查询点 $q(q\notin S)$，寻找 q 的最近邻近问题可以转化成寻找该点落入哪个 Voronoi 域中，因为 q 落入的 Voronoi 域中点 $p(\in S)$ 恰好是它的最近邻近点。另外，如果 $q\in S$，则与点 q 关联的 Voronoi 多边形 $V(q)$ 内的点就是所求 q 的最近邻近。这样就可以避免将 q 与平面上所有的点进行比较。在平面划分的内部定位一个点的问题叫做点定位。对于每个询问点 q，$O(\log n)$ 时间足以确定 q 所在的 Voronoi 域，即定位 q。因此寻找 q 的最近邻近问题在 $O(n\log n)$ 内可以解决。

2. 所有最近邻近

定义最近邻近图是一个无向图：它以 S 中的点为顶点，并且如果一个点是另一个点的最近邻近，那么它们之间有一条边连接。记最近邻近图为 NNG。

可以证明，$NNG\subseteq S$ 的 Delaunay 三角剖分。对于集合中的每个点寻找最近邻近的笨拙算法需要 $O(n^2)$ 时间，但是如果利用 S 的 Delaunay 三角剖分，则仅需搜索 Delaunay 三角剖分的 $O(n)$ 条边，并因此得到 $O(n\log n)$ 的复杂性。

4.4.2 最大化最小角的三角剖分

有限元分析技术经常用于分析复杂形状的结构性质，例如，汽车制造中车体模型的设

计。首先，将研究的域划分成有限元(子域)，然后用离散方法求解模拟结构力学的微分方程。该求解过程的稳定性依赖于划分的质量，即要寻找最大化最小角的三角剖分，也就是说要求所有三角形最大化最小角，这恰好是 Delaunay 三角剖分。另外，最小权三角剖分也可以满足要求，因为连线变短之后，三角形的内角(最小角)就变大。

设 T 是点集 S 的三角剖分，并且它的角序列$(\alpha_1,\alpha_2,\cdots,\alpha_{3t})$是从小到大排列的三角形角的表，其中 t 是 T 中三角形的个数。对于给定的点集 S，数 t 是一个常数。可以定义同一点集 S 两个三角剖分 T 和 T'之间的关系，利用该关系试图优化三角形的角，即如果 T 的角序列按字典序排列大于 T'的角序列，那么 $T\geqslant T'$(T 优于 T')：或者 $\alpha_1>\alpha_1'$，或者 $\alpha_1=\alpha_1'$，并且 $\alpha_2>\alpha_2'$，或者 $\alpha_1=\alpha_1'$，又 $\alpha_2=\alpha_2'$，$\alpha_3>\alpha_3'$ 等等。

可以证明，Delaunay 三角剖分 T 就角序列的优先关系而论是最大的：对于 S 的任何其他三角剖分 T'，有 $T\geqslant T'$成立。这表明 Delaunay 三角剖分最大化最小角。

4.4.3 最大空圆

给定平面上 n 个点的点集 S，寻找一个不包含 S 中点的最大圆，并且该圆的圆心在点集凸壳 CH(S)的内部。

定理 4-11 如果最大空圆的圆心 q 在 CH(S)的内部，那么 q 必然与 Voronoi 点重合。

证明 设点集 $S=\{p_1,p_2,\cdots,p_n\}$并在 CH(S)内任选一点 q，以 q 为圆心，$f(q)$为半径作圆，该圆内不包含 S 中的点。然后不断扩充该圆使其碰到 S 中一点，比如 p_1。由 p_1 出发过点 q 作射线 l，让 q 在射线 l 上移动到 q'，显然 $f(q')>f(q)$。当 q'在 CH(S)边界时，$f(q')$达到局部极大，如图 4-26 所示。

现设半径为 $f(q'')$时，该圆周通过点 p_2 和 p_3，$f(q'')$没有达到局部极大值。如果沿$\overline{p_2p_3}$的垂直平分线(即一条 Voronoi 边)移动 q''到 p，那么 $f(p)>f(q'')$，如图 4-26 所示。

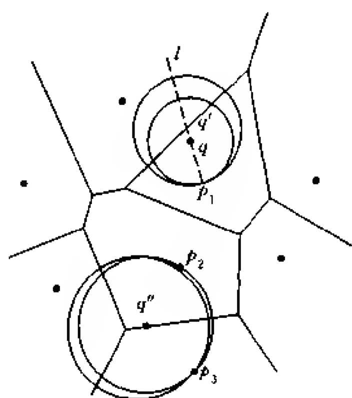


图 4-26 圆心在凸壳内部，圆周通过 1 个或 2 个点

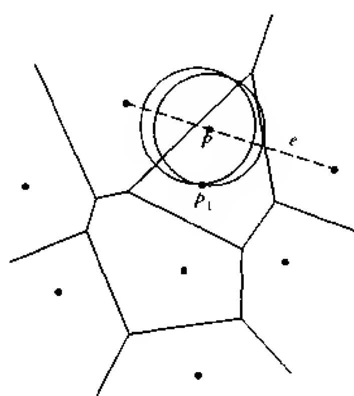


图 4-27 圆心在边 e 上，并且点 p_1 在圆周上的圆

只有当圆周通过 S 中三个点时，$f(p)$才能达到局部极大值。如果这三个点形成的三

角形包含点 p(圆心)，则 p 朝任意方向移动将导致离某个点更近，这样便使 $f(p)$减小。因此 p 不可能朝任意方向移动，p 只能是这三个点形成的三角形的外接圆圆心，所以定理结论成立。

定理 4-12 如果最大空圆的圆心 p 位于 BCH(S)上，那么 p 必定位于一条 Voronoi 边上。

证明 假设 p(圆心)在 BCH(S)上，并且 $f(p)$是极大的，以 p 为圆心的圆仅包含一个点 p_1。首先 p 不可能是凸壳 CH(S)的顶点，因为 CH(S)顶点是 S 中的点，这表明 $f(p)=0$，所以点 p 在 BCH(S)的一条边 e 上。然后点 p 沿 e 朝不同方向移动必增加它与 p_1 的距离，如图 4-27 所示。如果圆心在 p 的圆包含两个点 p_1 和 p_2。那么$\overline{p_1p_2}$的垂直平分线与 e 的交点作为圆心 p，$f(p)$是局部极大的。因此 p 位于一条 Voronoi 边上，定理成立。

显然，Voronoi 点是最大空圆圆心的候选点，但由于 Voronoi 点不一定在 CH(S)内部，所以只有在 CH(S)内部的 Voronoi 点才是最大空圆圆心的候选点。

确定点集 S 的最大空圆的算法如下。

步 1 计算 S 的 Voronoi 图 Vor(S)。

步 2 计算 S 的凸壳 CH(S)。max←0。

步 3 **for** 每个 Voronoi 点 v **do**

  if v 在 CH(S)内部

    then 计算 v 为圆心的圆的半径并修改 max。

步 4 **for** 每条 Voronoi 边 **do**

计算 CH(S)边 e 与 Voronoi 边的交点 p，计算圆心在 p 的圆的半径并修改 max。

步 5 输出 max。

算法中步 1 与步 2 均需要 $O(n\log n)$时间。由于 Voronoi 点的个数为 n，CH(S)的顶点数可能为 n，所以步 3 判定 Voronoi 点 v 是否在 CH(S)内耗费时间为 $O(n)$，判定 n 个 v 点则耗费为 $O(n^2)$。步 4 计算 e 与 Voronoi 边的交点需要时间 $O(n^2)$，因此算法的时间复杂性为 $O(n^2)$。但步 3、步 4 可以改进到在 $O(n\log n)$时间内完成，因此算法的时间复杂性为 $O(n\log n)$。

下面阐述周培德于 1995 年提出的一个求最大空圆的算法，该算法没有利用 Voronoi 图。

$Z_{4\text{-}4}$算法（求平面点集最大空圆的算法）

输入 域 $D=[0,A]^2$ 内的点集 $S=\{p_1,p_2,\cdots,p_n\}$。

输出 域 D 内不覆盖 S 中点的最大圆(圆心及半径)。

步 1 求点集 S 的凸壳顶点，设为 C_1，$C_1=\{p_1^1,p_2^1,\cdots,p_{m_1}^1\}$。

步 2 从 S 中删去 C_1，即 $S_1=S-C_1$。

步 3 求点集 S_1 的凸壳顶点，设为 C_2，$C_2=\{p_1^2,p_2^2,\cdots,p_{m_2}^2\}$。再从 S_1 中删去 C_2，得点集 S_2。依次重复，直至凸壳 C_i 内不包围 S 中的点或包围 S 中的 1 个点，2 个点及 3 个点，即 $S_i=\varnothing$或$|S_i|=1,2,3$。分别转步 4～步 7。

步 4 **while** $S_i=\varnothing$ **do**

步 4-1 对 C_i 的内域进行最小权三角剖分，得到 m_i-2 个三角形。

步 4-2 求 m_i-2 个三角形的外接圆，并排序外接圆半径 $r'_1, r'_2, \cdots, r'_{m_i-2}$，设 r'_1 最长。

步 4-3 依序检查这些圆是否包围 S 中的点，保留半径最长且不含 S 中点的圆。设该圆的圆心为 $o'(k')$，半径为 $r(k')$。转步 8。

步 5 **while** $|S_i|=1$ **do**

步 5-1 设 p 是 C_i 内的一个点。计算 p 与 C_i 各顶点的距离 $d(p,p'_j)(j=1,2,\cdots,m_i)$，并排序 $d(p,p'_1),d(p,p'_2),\cdots,d(p,p'_{m_i})$，其中 $d(p,p'_1)$ 最大。

步 5-2 **if** $d(p,p'_{j_1})\neq d(p,p'_{j_2}),(j_1,j_2=1,\cdots,m_i,j_1\neq j_2)$

then 连接 p 与 p'_{m_i}，p 与 p'_{m_i-1}，$\cdots$，p 与 p'_{m_i-k}，使剩余点 $p'_1,p'_2,\cdots,p'_{m_i-k+1},p$ 成一凸壳，$(k=0,1,2,\cdots)$，记为 C'_i。C'_i 内不含 S 中的点，转步 5-3。

else 按 C_i 的边长排序：$\overline{p'_1p'_2},\overline{p'_2p'_3},\cdots,\overline{p'_{m_i-1}p'_{m_i}},\overline{p'_{m_i}p'_1}$。分别求出三角形 $pp'_1p'_2$，三角形 $pp'_2p'_3$，…的外接圆。设 $o'(k')$ 为圆心，$r(k')$ 为半径的圆（圆内不含 S 中的点）最大，转步 5-4。

步 5-3 用步 4 的方法求 C'_i 内不覆盖 S 中点的最大圆，并与三角形 $pp'_{m_i}p'_{m_i-1}$，三角形 $pp'_{m_i}p'_{m_i-2}$，…的外接圆比较。设 $o'(k')$ 为圆心，$r(k')$ 为半径的圆（圆内不含 S 中的点）最大。

步 5-4 保留 $o'(k')$，$r(k')$。转步 8。

步 6 **while** $|S_i|=2$ **do**

步 6-1 设 p_1,p_2 是 C_i 内的两个点。连接 p_1 与 p_2，位于 $\overrightarrow{p_1p_2}$ 右（左）侧的 C_i 顶点，记为 $C_1(C_2)$。$C_1(C_2)$ 与 p_1,p_2 构成凸壳 $C'_1(C'_2)$。用步 4 的方法求 $C'_1(C'_2)$ 中不覆盖 S 中点的最大圆，并与三角形 $p_1p'_jp'_{j+1}$，三角形 $p_2p'_{j_1}p'_{j_1+1}(p'_j,p'_{j_1+1}\in C_2;p'_{j+1},p'_{j_1}\in C_1)$ 的外接圆比较。设 $o'(k')$ 为圆心，$r(k')$ 为半径的圆（圆内不含 S 中的点）最大。

步 6-2 保留 $o'(k')$，$r(k')$。转步 8。

步 7 **while** $|S_i|=3$ **do**

步 7-1 设 p_1,p_2,p_3 是 C_i 内的 3 个点。顺序连接 3 点成一个三角形 $p_1p_2p_3$。用步6-1 的方法求环域 C_i－三角形 $p_1p_2p_3$ 内不覆盖 S 中点的最大圆，保留求得的圆心及半径。

步 7-2 求三角形 $p_1p_2p_3$ 的外接圆，与步 7-1 求得的半径比较。保留半径最长者（记为 $r(k')$）及相应的圆心 $o'(k')$。转步 8。

步 8 用步 7 的方法求环域（C_{i-1} 与 C_i 之间）的最大空圆，得圆心 $o''(k)$ 及半径 $r'(k)$。重复步 8，直至 $i=1$。

if $r'(k)>d(o''(k),l_1)$，l_1 是 D 的边界

then 删去 $o''(k)$，以长度较小的半径代替 $r'(k)$。

步 9 求 $R(k)=\max(r(k'),r'(k))$ 及相应的圆心 $o(k)$。输出 $R(k)$ 与 $o(k)$。

$Z_{4\text{-}4}$ 算法的步 1 求凸壳 C_1（设顶点数为 m_1），最多需要 $O(n\log n)$ 次比较与 $O(n)$ 次乘法。步 2 用 n 次比较可以求得点集 S_1。步 3 要反复执行 b 次，每执行一次求得一层凸壳。设每层凸壳的顶点数呈递减序列。不妨设该序列为 $m_1,\cdots,5,4$（共 b 个数，$b<n/3$），有

$\sum_{i=4}^{m_i} i \leqslant n$。因此步 3 所需要的比较次数为

$$n\log n + (n - m_1)\log(n - m_1) + \cdots + 4\log 4$$
$$\leqslant [n + (n - m_1) + \cdots + 4]\log n = O(n^2\log n)$$

乘法次数为

$$n + (n - m_1) + \cdots + 4 = O(n^2)$$

如果顶点数序列不是以形式 $m_1,\cdots,5,4$ 出现，则步 3 的复杂性的阶仍是上述估计的值。

步 4-1 最多需要 $O(n^3/m_i^3)$ 次乘法和 $O(n^3/m_i^3)$ 次比较，其中 m_i 是最内层凸壳顶点数。步 4-2 耗费 $14(m_i-2)$ 次乘法，$(m_i-2)\log(m_i-2)$ 次比较。步 4-3 的耗费不超过 $2n$ 次乘法和 n 次比较。因此步 4 耗费的乘法次数为

$$O(n^3/m_i^3) + 14(m_i - 2) + 2n = O(n^2)$$

比较次数为

$$O(n^3/m_i^3) + (m_i - 2)\log(m_i - 2) + n = O(n^2)$$

步 5～步 7 的复杂性的限界均不超过步 4 的复杂性的阶。

步 8 所需要的乘法次数不超过 $O(n^2)$，比较次数不超过 $O(n^2)$。步 9 耗费 b 次比较，即不超过 $n/3$ 次比较便可以求得点集 S 的最大空圆。总之，$Z_{4,4}$ 算法需要的乘法次数为

$$O(n) + O(n^2) + O(n^2) + O(n^2) = O(n^2)$$

比较次数为

$$O(n\log n) + n + O(n^2\log n) + O(n^2) + O(n^2) + n/3 = O(n^2\log n)$$

4.4.4 最小生成树

点集 S 的最小生成树(MST)是连接 S 中所有点的最小长度的树，即最小生成树的结点恰好是 S 中的点。两结点之间的连线以 Euclidean 长度度量时，该树称为 Euclidean 最小生成树(EMST)。图 4-28 所示是一个例子。MST 有许多应用，例如，许多局域网络利用树的形式生成基结点，MST 是最小化总线路长度的网络拓扑。

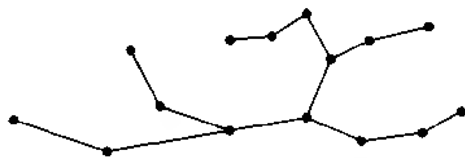


图 4-28 一棵 Euclidean 最小生成树

考虑计算平面点集的 MST 问题，该问题可以转化为计算平面完全图 G 的 MST 问题，解决后者的一种方法是基于贪心思想设计的。该方法是不断添加还没有选取的最短边至树中，同时保持树(不含回路)的特性。这个算法称为 Kruskal 算法，描述如下：

步 1 按长度从小到大对所有边排序得：$e_1,e_2,\cdots,e_{\frac{n(n-1)}{2}}$

步 2 $T\leftarrow\varnothing, i\leftarrow 1, j\leftarrow 1$

步 3 **while** $j<n$ **do**

if $T+e_i$ 是树(非回路)

  then $T \leftarrow T+e_i, j \leftarrow j+1$

$i \leftarrow i+1$

Kruskal 算法中,符号 $T+e_i$ 表示树 T 与边 e_i 的并,而步 1 决定该算法的时间复杂性为 $O(|E|\log|E|)$,其中 $|E|$ 是图 G 的边数目。

由于图 G 有 $\frac{n(n-1)}{2}$ 条边,因此分类的复杂性是 $O(n^2\log n)$,故求解 MST 需要 $O(n^2\log n)$ 时间。如果用 Delaunay 三角剖分边的集合代替完全图的边集合,即利用 Delaunay 三角剖分边来构造 MST,那么将使 Kruskal 算法的复杂性得到改进。

定理 4-13 最小生成树 MST 是 Delaunay 三角剖分 $DT(S)$ 的一个子集:$\mathrm{MST} \subseteq DT(S)$。

证明 要证明如果 $\overline{ab} \in \mathrm{MST}$,则 $\overline{ab} \in DT(S)$。假设 $\overline{ab} \in \mathrm{MST}$,但 $\overline{ab} \notin DT(S)$,那么通过说明所假定的 MST 不是最小的来获得矛盾。

如果 $\overline{ab} \in DT(S)$,则由定理 4-9,过 a,b 有一个空圆。因此如果 $\overline{ab} \notin DT(S)$,那么过 a,b 的圆不可能是空的。也就是说,具有直径 $\overline{ab}$ 的圆周上或圆内必有 S 中的点,假设 c 在该圆周上或者圆内,如图 4-29 所示,那么 $|\overline{ac}| < |\overline{ab}|$,并且 $|\overline{bc}| < |\overline{ab}|$;即使 c 在圆周上这些不等式也成立,因为 c 不同于 a 和 b,删去 $\overline{ab}$ 将把树 T 分成两棵树 T_a 和 T_b。$a \in T_a$,$b \in T_b$。不失一般性,设 $c \in T_a$。删去 $\overline{ab}$ 并且添加边 $\overline{bc}$ 构成一棵新树 T',$T' = T_a + \overline{bc} + T_b$。树 T' 的长度更短。因此包含边 $\overline{ab}$ 的树的长度不可能最小。这样,由否定 $\overline{ab}$ 在 $DT(S)$ 中,推得 MST 不是最小生成树,矛盾。 证毕。

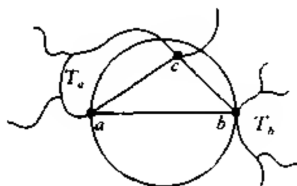
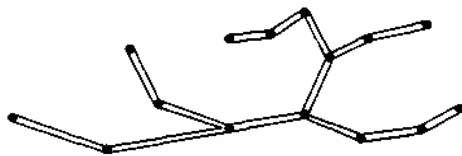


图 4-29 $T_a+\overline{ab}+T_b$ 的长度比 $T_a+\overline{bc}+T_b$ 长

改进 Kruskal 算法如下:首先在 $O(n\log n)$ 时间内构造 Delaunay 三角剖分,然后耗费 $O(n\log n)$ 时间分类 $O(n)$ 条边,Kruskal 算法的其余部分不修改,并且在 $O(n\log n)$ 时间内可以完成。因此构造点集 S 的 MST 的时间复杂性为 $O(n\log n)$。

4.4.5 货郎担问题

借助 Delaunay 三角剖分与最小生成树可以设计求解货郎担问题(英文缩写为 TSP)的一种近似算法,其思想是首先寻找点集的 MST,如图 4-28 所示,然后沿 MST 来回走两遍,最后用边替代的方法消去重复的边,如图 4-30 所示。


图 4-30 MST 长度的两倍

用上述方法得到的回路全长 T 不会超过 MST 长度的两倍,因而小于最短回路长度

的 2 倍。即 $T<2T_1$，其中 T_1 为 TSP 的最短回路长度。

可以用其他探索法改进上述结果，其中 $T<\frac{3}{2}T_1$ 是最好的结果。

4.4.6 中轴

在某种程度上，从多边形 P 的中轴结构可以看出形状的特征，中轴已在模式识别和计算机视觉中得到应用。

定义多边形 P 的中轴为 P 内的点集，该点集中的点与 ∂P 不同边（或多边形边的延长线）中两个或两个以上点距离相等。也就是说，与 ∂P 不同边（或多边形边的延长线）中两个或两个以上点等距离的点的轨迹定义为多边形 P 的中轴。

如果多边形 P 是一矩形，则其中轴如图 4-31 所示。图中矩形内水平线上的点与矩形的上下边界的垂直距离相等，而对角线上的每个点与矩形的两条相邻的边的距离相等，并且矩形内水平线段的两个端点与矩形的 3 条边距离相等。

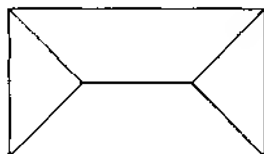


图 4-31　矩形的中轴

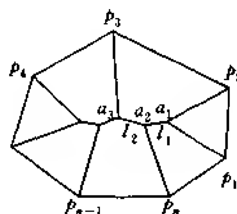


图 4-32　凸多边形的中轴

一个较复杂的例子如图 4-32 所示，这是 7 个顶点的凸多边形。由该例子可以看出凸多边形 P 的中轴是一棵树 T，它的叶是 P 的顶点，而内部结点是与 ∂P 的 3 条边相切的圆的圆心。对于非凸多边形也是如此，中轴上的每个点是圆心，该圆至少与两条边相切，正如 Voronoi 点是过 3 个点的圆的圆心一样。

下面介绍求凸多边形中轴的一种算法。设 p_i 与 p_{i+1} 是凸多边形的相邻顶点，并且 p_i 又表示顶点角，$i=\overline{1,n}$。

$Z_{4\text{-}5}$算法（求凸多边形中轴的算法）

步 1　作各顶点角 p_i 的平分角线。

步 2　求顶点角 p_i 与 p_{i+1} 分角线的交点及交点至相关边（即 $\overline{p_ip_{i+1}}$）的距离，设为 d_1，$d_2\cdots$，d_n。

步 3　计算 $d=\min(d_1,d_2,\cdots,d_n)$，设 $d=d_1$。a_1 表示 p_1 与 p_2（顶点按逆时针方向重新排序）两顶点角分角线的交点，如图 4-32 所示。

步 4　$i\leftarrow 1$

步 5　作 $\overline{p_np_1}$ 与 $\overline{p_{i+2}p_{i+1}}$ 夹角的分角线 l_i，l_i 必经过 a_i。

步 6　作顶点角 p_n 的分角线，与 l_i 交于 a_{i+1}。

步 7　作 $\overline{p_{n-i}p_{n-i+1}}$ 与 $\overline{p_{i+2}p_{i+1}}$ 夹角的分角线 l_{i+1}，l_{i+1} 必经过 a_{i+1}。

步 8　作顶点角 p_{i+2} 的分角线，与 l_{i+1} 交于 a_{i+2}。

步 9 作$\overline{p_{n-i}p_{n-i+1}}$与$\overline{p_{i+3}p_{i+2}}$夹角的分角线 l_{i+2}，l_{i+2}必经过 a_{i+2}。

步 10 作顶点角 p_{n-i}的分角线，与 l_{i+2}交于 a_{i+3}。

步 11 循环执行步 9 与步 10，执行步 9 时按减序、增序轮换改变$\overline{p_{n-i}p_{n-i+1}}$、$\overline{p_{i+3}p_{i+2}}$的下标；顶点角下标分别按增序、减序交替执行步 10。直至执行步 9 时两线段的夹角为顶点角。

步 12 输出折线 $a_1a_2\cdots a_k$ 及折线各顶点与相应凸多边形顶点的连线。

图 4-32 已示出 $Z_{4\text{-}5}$算法的求解过程。当$|\overline{a_3a_1}|<|\overline{a_2a_1}|$时，算法中的执行顺序将有改变。步 1 与步 2 分别耗费线性时间，步 3 需要 $n-1$ 次比较，步 5 至步 10 分别需要常数时间，步 11 至步 9 的循环不超过 n 次，因此该算法的时间复杂性为 $O(n)$。

当凸多边形为矩形时，计算各顶点角平分线的交点，连接短边相关交点即得中轴，如图 4-31 所示。

利用凸壳及其中轴，可以设计出求解 TSP 的近似算法，下面描述该算法。

$Z_{4\text{-}6}$算法(求 TSP 的算法)

步 1 计算点集 S 的凸壳 C，设 C 的顶点集为 $C(1)=\{q_1^1,q_2^1,\cdots,q_{m_1}^1\}$，$C$ 的边集为 $E(1)=\{e_1^1,e_2^1,\cdots,e_{m_1}^1\}$。

步 2 利用 $Z_{4\text{-}5}$算法计算凸多边形 C 的中轴，该中轴划分点集 S 为 m_1 个子点集，设为 $S_1,S_2,\cdots,S_{m_1}$，$S=(\bigcup_{j=1}^{m_1}S_j)\cup C(1)$。

步 3 $i\leftarrow 1$，$S(i)\leftarrow S$。

步 4 $S(i+1)\leftarrow S(i)-C(i)$。

步 5 计算点集 $S(i+1)$的凸壳，记凸壳的顶点集为 $C(i+1)$。

步 6 $i\leftarrow i+1$，重复执行步 4 与步 5，直至 $S(i+1)=\varnothing$。设最内层凸壳的顶点集为 $C(k)=\{q_1^k,q_2^k,\cdots,q_{m_k}^k\}$，边集 $E(k)=\{e_1^k,e_2^k,\cdots,e_{m_k}^k\}$。

步 7 分别计算点 $q_1^k,q_2^k,\cdots,q_{m_k}^k$至点 $q_1^{k-1},q_2^{k-1},\cdots,q_{m_{k-1}}^{k-1}$及点 $q_1^{k-2},q_2^{k-2},\cdots,q_{m_{k-1}}^{k-2}$的距离，并依最小与次最小距离值将点 $q_1^k,q_2^k,\cdots,q_{m_k}^k$归到相应的子点集 $S_1,S_2,\cdots,S_{m_1}$，记为 $S_{1,2}^1,S_{2,3}^1,\cdots,S_{m_1,1}^1$，其中 $S_{l,l+1}^1$是与边 e_l^1 相关的子点集，$l=\overline{1,m_1}$，$S_{m_1,m_1+1}^1=S_{m_1,1}^1$。对中轴上及附近点类似处理。

步 8 求通过 $S_{l,l+1}^1$中各点的子路径：

步 8-1 $S_{l,l+1}^1(1)\leftarrow S_{l,l+1}^1$，$l=\overline{1,m_1}$

步 8-2 $j\leftarrow 1$

步 8-3 $l\leftarrow 1$

步 8-4 $S_{l,l+1}^1(j)\leftarrow S_{l,l+1}^1(1)\cup\{q_l^1,q_{l+1}^1\}$

步 8-5 计算 $S_{l,l+1}^1(j)$的凸壳，设凸壳为 $A_{l,l+1}^1(j)$，该凸壳由原有边(比如$\overline{q_l^1q_{l+1}^1}$)和新边组成，删去原有边得到凸壳新边集 $A_{l,l+1}'^1(j)$(比如 $A_{l,l+1}'^1(j)=A_{l,l+1}^1(j)$的边集$-\overline{q_l^1q_{l+1}^1}$)，由 $S_{l,l+1}^1(j)$删去 $A_{l,l+1}^1(j)$的顶点集得到 $S_{l,l+1}'^1(j)$。设新边集规模$|A_{l,l+1}'^1(j)|=m_l(j)$。

步 8-6 依 $S_{l,l+1}'^1(j)$中点到 $A_{l,l+1}'^1(j)$各边距离的最小值划分 $S_{l,l+1}'^1(j)$中点为 $m_l(j)$个子集，设为 $S_{l,l+1}'^1(j,1)$，$S_{l,l+1}'^1(j,2)$，$\cdots$，$S_{l,l+1}'^1(j,m_l(j))$。

步 8-7 $l \leftarrow l+1$，重复步 8-4 至步 8-6，直至 $l=m_1+1$。

步 8-8 $j \leftarrow j+1$，以 $S'^{j}_{l,l+1}(j-1,u)(u=\overline{1,m_l(j-1)})$分别代替 $S^j_{l,l+1}(j-1)$，重复执行步 8-3 至步 8-7，直至所有子点集为空。得到 m_1 条子路径 $L_l, l=\overline{1,m_1}$。

步 9 各子路径 L_l 在 $q^1_l(l=\overline{1,m_1})$处连接成一条回路 H。

步 10 在 H 中检查相邻 4 点(比如 a,b,c,d)组成的路径是否是最短的路径：比如，在 $a\to b\to c\to d$ 和 $a\to c\to b\to d$ 中选一条短的路径并要求$\overline{ac}$与$\overline{bd}$不相交。

步 11 计算回路长度并输出结果。

$Z_{4\text{-}6}$算法的步 1 至步 7 将点集 S 划分为 m_1 个子点集，利用步 8 中的方法寻求通过子点集中各点的子路径。步 10 进行局部优化，从而使路径进一步缩短。

$Z_{4\text{-}6}$算法的第 1 步求点集的凸壳，需要 $O(n\log n)$时间。第 2 步计算凸壳的中轴耗费时间为$O(n)$，此外划分点集S 成 m_1 个子集 $S_1,S_2,\cdots,S_{m_1}$，需要$O(nm_1)$时间。第 3 步只用常数时间。第 4 步至第 6 步逐层求凸壳，其复杂性不超过 $O(kn\log n)$，其中 k 为凸壳的层数。第 7 步重新划分最内层凸壳顶点，设最内层凸壳顶点数为 m_k，第 $k-1$ 层凸壳顶点数为 m_{k-1}，则第 7 步的时间耗费为 $m_k \cdot m_{k-1} < n^2$。第 8 步求通过子点集 $S^j_{l,l+1}$中各点的子路径，其中步 8-1 至步 8-4 耗费常数时间。如果最外层凸壳有 m_1 条边，点集 S 被分成 m_1 个子点集，每个子点集有$\frac{n-m_1}{m_1}=w$ 个点，则步 8-5 需要的时间为 $O(w\log w)$，步 8-6 的时间耗费不超过 $O(w^2)$。步 8-7 循环的时间复杂性不超过 $O(m_1w^2)$。步 8-8 循环的开销为 $O(\lceil \log w\rceil \cdot m_1 \cdot w^2)$，即步 8 的时间复杂性为 $O\left(m_1 \cdot \frac{(n-m_1)^2}{m_1^2} \cdot \log\frac{n-m_1}{m_1}\right) \leqslant O(n^2\log n)$。步 9 与步 11 所需时间可以不计，步 10 耗费线性时间。因此 $Z_{4\text{-}6}$算法总的时间耗费为

$$O(n\log n)+O(n)+O(nm_1)+O(kn\log n)+O(n^2)+O(n^2\log n)+O(n)=O(n^2\log n)$$

将 $Z_{4\text{-}6}$算法应用于中国 31 个省会城市，得到一条长度为 15404km 的回路，与运用分支定界法所求得的路径长度完全一致。图 4-33 显示 $Z_{4\text{-}6}$算法不同步骤所得到的结果。这个算法是一个近似算法，但对于中国 31 个城市的货郎担问题来说，它是一个多项式时间的精确算法。

将 $Z_{4\text{-}5}$算法作些修改便可以用于求一般简单多边形的中轴：首先确定多边形的凸凹顶点(并划分成凸部分)，然后对于凸点用 $Z_{4\text{-}5}$算法思想处理，而对于凹点，要考虑凹点关联的两条边与多边形另一条边构成的夹角分角线的交点，例如图 4-34(d)中点 q_2；凹点关联的两条边分别与另一凹点关联的两条边构成的夹角分角线的交点，例如图 4-34(c)中点 q_3；凹点关联的两条边分别与多边形另一凸点关联的两条边构成的夹角分角线的交点，它位于该凸角分角线上，例如图 4-34(a)中点 q_2 与点 q_4，等等。

$Z_{4\text{-}7}$算法(简单多边形 P 的中轴)

步 1 计算 P 的凸、凹顶点，设 $p_1,p_2,\cdots,p_k$ 是凸点，$p'_1,p'_2,\cdots,p'_l$ 是凹点，$k+l=n$。

步 2 对 $p_i(i=\overline{1,k})$利用 $Z_{4\text{-}5}$算法中步 1 至步 3 选择中轴非叶结点的起点。设起点为 q_1(可能有多个起点)，它是凸顶点角 p_i 与 p_{i+1}分角线的交点。或者 p_i 与 p_{i+1}之间有一个凹点 p'_i，此时计算凸顶点角的分角线 l_i 与 l_{i+1}、p_i 角一边的延长线与 p_{i+1}角一边的夹角的

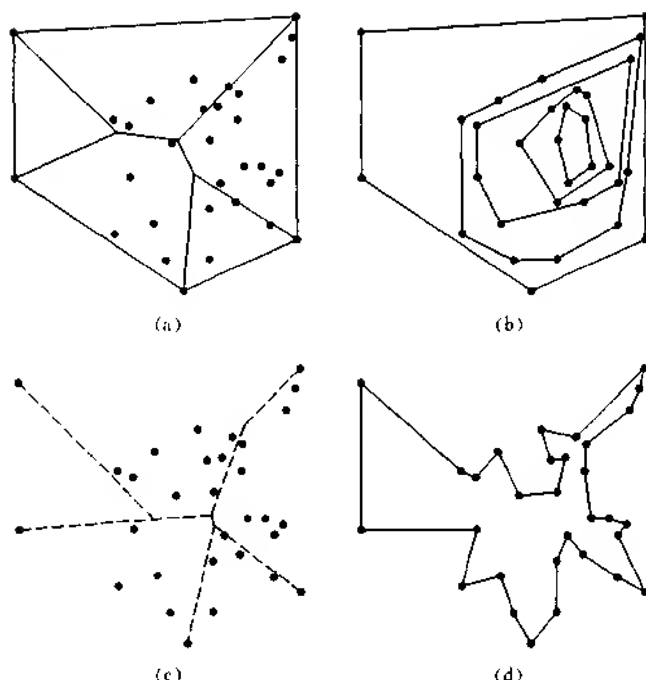


图 4-33 $Z_{4\text{-}6}$算法执行过程的图示

(a) 执行步 1、步 2 所得到的结果 (b) 执行算法步 3 至步 6 的结果

(c) 执行步 7 的结果 (d) 执行步 8、步 9 与步 10 的结果

分角线分别与 l_i,l_{i+1}交于 q_1 和 q_2,以 q_2 为起点,如图 4-34(f)。

步 3 如果 p_i 与 p_{i+1}之间没有其他顶点,则计算$\overline{p_{i-1}p_i}$延长线与$\overline{p_{i+1}p_{i+2}}$延长线夹角的分角线,设为 l_1,l_1 通过 q_1,如图 4-34(a)所示。

步 4 如果 p_{i-1}与 p_{i+2}是凸点并且它们之间没有凹点,则用 $Z_{4\text{-}5}$算法中步 4 至步 11 计算中轴的非叶结点。

否则,p_{i-1}是凹点,p_{i+2}是凸点,则计算$\overline{p_{i-2}p_{i-1}}$延长线与$\overline{p_{i+2}p_{i+3}}$延长线夹角的分角线 l,l 与 l_1 交于 q_2,如图 4-34(a)所示。

否则,p_{i-1}和 p_{i+2}是凹点,则计算$\overline{p_{i-2}p_{i-1}}$延长线与$\overline{p_{i+2}p_{i+3}}$延长线夹角的分角线 l,l 与 l_1 交于 q_2,如图 4-34(b)所示。

否则,p_{i-1}是凹点,p_{i+2}是凸点,而 p_{i+3}是凹点,角 p_{i+2}的分角线与 l_1 交于 q_2,则计算$\overline{p_ip_{i-1}}$延长线与$\overline{p_{i+2}p_{i+3}}$延长线夹角的分角线 l,l 经过 q_2,如图 4-34(c)所示。

否则,p_{i+2}是凸点,p_{i+3}是凹点,则计算$\overline{p_{i+1}p_{i+2}}$与$\overline{p_{i+2}p_{i+3}}$夹角分角线 l,l 与 l_1 交于 q_2,如图 4-34(d)所示。

否则,p_{i-2},p_{i-1}是凸点,p_{i+2},p_{i+3}是凹点,则计算$\overline{p_{i-1}p_i}$延长线与$\overline{p_{i+3}p_{i+2}}$延长线夹角分角线 l_1,l_1 经过 q_1;顶点角 p_{i-1}的分角线与 l_1 交于 q_2;$\overline{p_{i-1}p_{i-2}}$延长线与$\overline{p_{i+2}p_{i+3}}$延长线夹角分角线 l,l 经过 q_2,如图 4-34(e)所示。

步 5 重复步 4,直至计算完所有顶点。

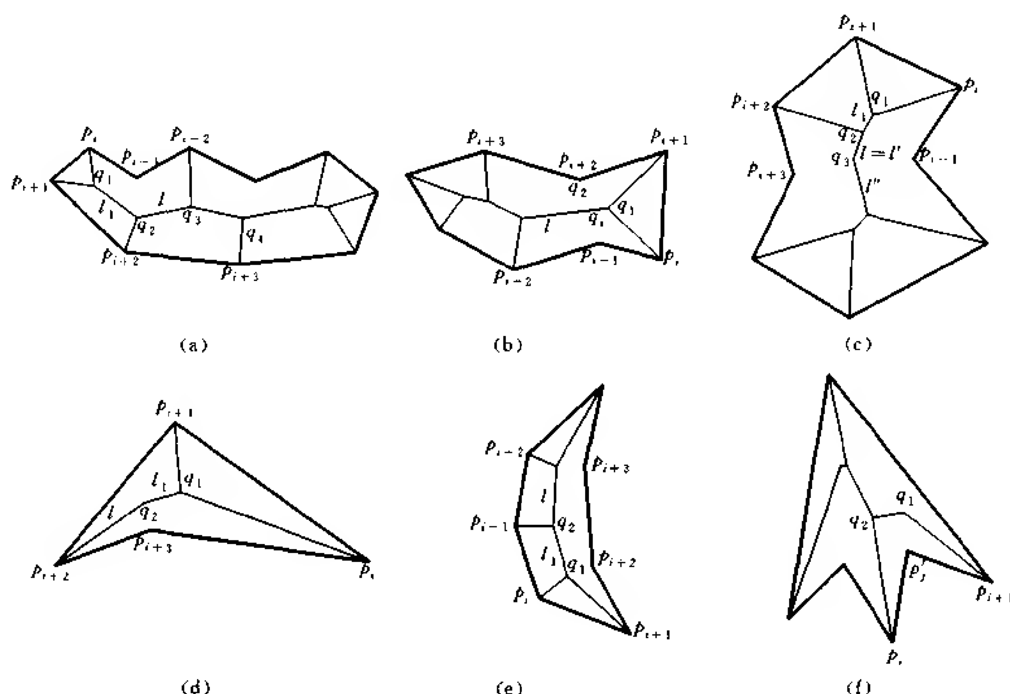


图 4-34　简单多边形的中轴

图 4-34 中已示出 $Z_{4\text{-}7}$算法的执行过程，图中列出 6 种基本情况，由它们可以组合成各种复杂的简单多边形。另外，图 4-34(c)中，如果角 p_{i-1}、p_{i+3}的边的延长线组成的角的平分角线 l'，l''不相交或不交于延长线组成的四边形内部(或延长线没有组成四边形)，则计算 p_{i-1}的一条边与 p_{i+3}的一条边的延长线夹角的分角线，该分角线与 l'，l''相交，组成部分中轴。然后，逐条边递增计算。最后得到原多边形的中轴。

$Z_{4\text{-}7}$算法中步 1 耗费线性时间，步 2、步 3 和步 4 均耗费常数时间，步 5 的循环次数不超过 n，因此 $Z_{4\text{-}7}$算法的时间复杂性为 $O(n)$，优于 Lee(1982)提出的复杂性为 $O(n\log n)$的算法。将 $Z_{4\text{-}7}$算法用于图 4-35，得到该图所示的中轴。

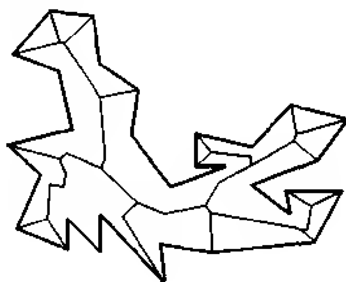


图 4-35　$Z_{4\text{-}7}$算法应用举例

4.4.7 Voronoi 图与凸壳的关系

1986 年 Edelsbrunner 和 Seidel 发现了 Delaunay 三角剖分与较高维凸壳之间的关系。下面首先介绍一维 Delaunay 三角剖分与二维凸壳之间的关系，然后将此关系推广到二维 Delaunay 三角剖分和三维凸壳。研究这种关系有助于它们之间的相互转换。

1. 一维 Delaunay 三角剖分

设 $S=\{x_1,\cdots,x_n\}$ 是 x 轴上点的集合，显然一维 Delaunay 三角剖分是连接 x_1 与 $x_2,\cdots,x_n$ 的路径。这可以看成是坐标为 (x_i,x_i^2) 的二维点集合在 x 轴上的投影，另外也可以看成是由 x_i 向上到抛物线 $z=x^2$ 的投影，只要删去凸壳的顶边，这些二维点的凸壳向下投影便得到一维 Delaunay 三角剖分。

在点 $x=a$ 处，抛物线 $z=x^2$ 的斜率是 $2a$，因此，在点 (a,a^2) 处抛物线的切线方程为

$$z-a^2=2a(x-a)$$

$$z=2ax-a^2$$

当切线垂直平移距离 r^2 时，讨论该切线与抛物线之间的交。切线上升 r^2 时，切线方程为

$$z=2ax-a^2+r^2$$

求切线与抛物线的交点如下

$$z=x^2=2ax-a^2+r^2$$

$$x=a\pm r$$

因此上升的切线与抛物线在 $x=a\pm r$ 处相交。值得注意的是，$x=a\pm r$ 可以看成是圆心在 a 半径为 r 的一维圆的方程，如图 4-36 所示，图中 $a=5$，$r=3$，圆是线段[2,8]。

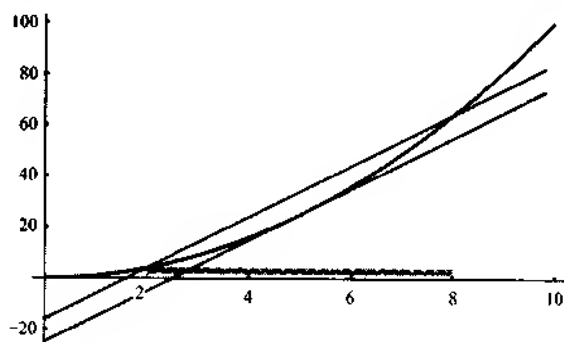


图 4-36　$a=5$ 时，切线方程是 $z=10x-25$

2. 二维 Delaunay 三角剖分

设抛物面是 $z=x^2+y^2$，如图 4-37 所示。在 xy 平面上给定点，并向上投影到抛物面，即映射：

$$(x_i,y_i)\rightarrow(x_i,y_i,x_i^2+y_i^2)$$

给定三维点集的凸壳，如图 4-38 所示，删去该壳的"上壳"面，即小面法向量与 z 轴正

向量点积为正的那些小面，剩下"下壳"面，将"下壳"面投影到 xy 平面，便得到 Delaunay 三角剖分，如图 4-39 所示。

下面推导三维凸壳与二维 Delaunay 三角剖分之间的关系。

点 (a,b) 处切平面方程是

$$z = 2ax + 2by - (a^2 + b^2)$$

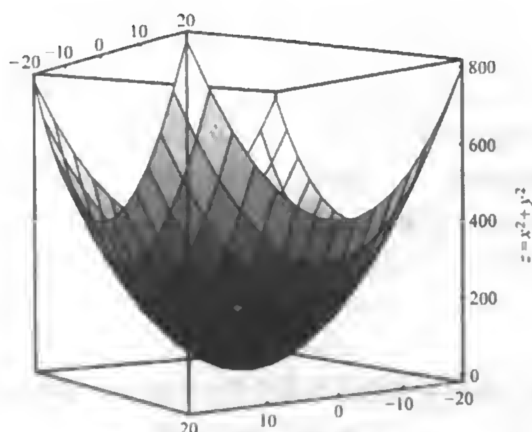
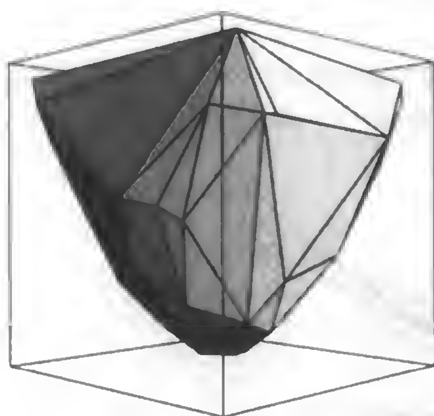


图 4-37　点向上投影到抛物面



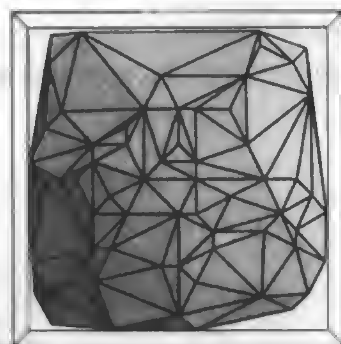


图 4-38　向上投影到抛物面的 65 个点的凸壳　　　图 4-39　从 $z=-\infty$ 所看到的抛物面壳

将该切平面向上平移 r^2，得到

$$z = 2ax + 2by - (a^2 + b^2) + r^2$$

由下式可以求出平移的切平面与抛物面的交

$$z = x^2 + y^2 = 2ax + 2by - (a^2 + b^2) + r^2$$

$$(x - a)^2 + (y - b)^2 = r^2$$

即平移的切平面与抛物面的交是椭圆，该椭圆在 xy 平面上的投影是一个圆（圆心在点 (a,b)，半径为 r），如图 4-40 与图 4-41 所示。

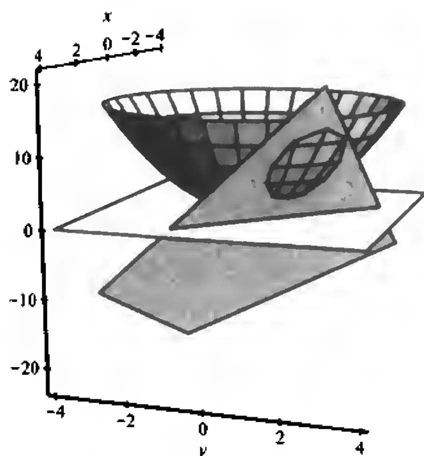


图 4-40 $(a,b)=(2,2),r=1$，切割抛物面的切平面

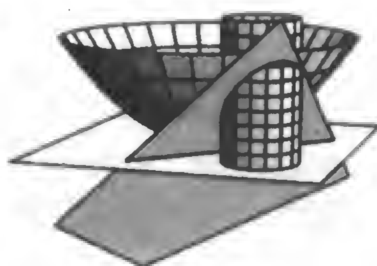


图 4-41 图 4-40 中的椭圆投影到 xy 平面上是半径为 1 的圆

设平面 π 是三维凸壳一小面所在的平面，并且 π 由抛物面$\triangle=(p_i,p_j,p_k)$上 3 点确定，平面 π 切割抛物面。如果垂直向下平移 π，它将与抛物面在(a,b,a^2+b^2)处相切，那么 π 可以看成是切平面 τ 向上平移 r^2 得到的。

因为抛物面的其他所有点在 π 的上方，当然也在切平面 τ 的上方，所以这些点投影到 xy 平面上半径 r 的圆的外部，因此 π 切割抛物面的切口在 xy 平面上的投影是一个空圆（不含抛物面其他点的投影点）。这样，凸壳的一小面投影到 xy 平面上便形成一个 Delaunay 三角形，"下壳"的每个三角形小面对应于一个 Delaunay 三角形，因此"下壳"投影到 xy 平面上得到 Delaunay 三角剖分，如图 4-39 所示。

综上所述，二维点集的 Delaunay 三角剖分恰好是三维点集的"下壳"到 xy 平面上的投影，反之，将二维点集的 Delaunay 三角剖分向上映射到抛物面 $z=x^2+y^2$ 可以得到三维点集的"下壳"。

因为三维中的凸壳可以在 $O(n\log n)$时间内计算，这意味着二维 Delaunay 三角剖分可以在相同的时间限界内求得。一旦进行完 Delaunay 三角剖分，就比较容易计算 Voronoi 图，这是构造 Voronoi 图的另一种 $O(n\log n)$算法。

事实上，Voronoi 图与高维中凸壳之间的关系在任意维中都成立。因此，三维中的 Voronoi 图可以由四维中的凸壳来构造，一般来说，d 维点集的 Voronoi 图的对偶图是 $d+1$维中点集"下壳"的投影。

4.4.8 Voronoi 图的推广

平面上 n 个点的点集 $S=\{p_1,p_2,\cdots,p_n\}$的 Voronoi 图是平面域的一个划分，该划分产生的每个子域 $V(p_i)$是具有下述性质的点的轨迹：子域内的点 $q(q\neq p_i)$与 p_i 的距离小于 S 中其他点与 q 的距离，即 $d(q,p_i)<d(q,p_j),p_j\in S,j=\overline{1,n},j\neq i$。在上述描述中，平面、点集与点 p_i 是 3 个要素，对这 3 个要素可以进行推广。比如，保持要素"平面"不变，即仍在二维中，而把点集推广到包含其他几何对象，例如线段、圆等；给定平面上 n 个点，寻

找距 n 个点中的 k 个点最近邻近的点的轨迹，该轨迹称为 k 阶 Voronoi 图，记为$\mathrm{Vor}_k(S)$。当 k 为 $n-1$ 时，便是另一种最远点意义下的 Voronoi 图，比如 $V(p_i)$表示离 p_i 最远点的轨迹，即对于这些点来说 p_i 是最远点。

定义 k 阶 Voronoi 图为 S 的 k 个元素的子集的所有广义 Voronoi 多边形的集合，表示如下：

$$\mathrm{Vor}_k(S) = \bigcup V(T), T \subset S, |T| = k$$

其中 $V(T)=\{p \mid \forall\ v \in T, \forall\ w \in S-T, d(p,v)<d(p,w)\}$。当 $k=2$ 时便是二阶 Voronoi 图。图 4-42 示出 8 个点的二阶 Voronoi 图，其中点对(1,5)，(5,7)等都没有相应的 Voronoi 多边形，点对(4,6)相应的 Voronoi 多边形是最接近点对(4,6)的点的轨迹。本章前面介绍的 Voronoi 图称为一阶 Voronoi 图。下面只是阐述二阶 Voronoi 图。

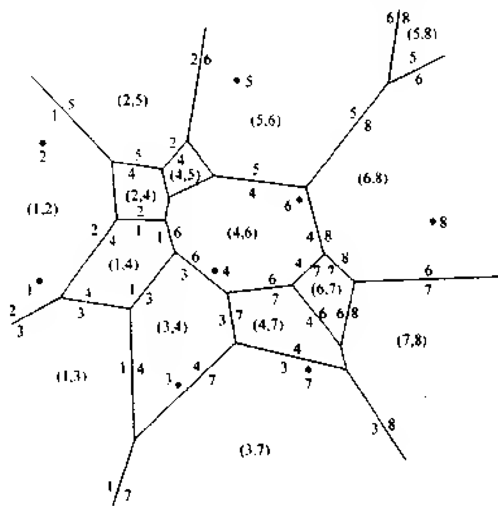


图 4-42　二阶 Voronoi 图

在图 4-42 中，只是 15 个点对(1,2)，(1,3)，(1,4)，(2,4)，…，(7,8)等有相应的 Voronoi 多边形，这就是说，某些 $V(T)$可能是空的，比如(1,5)，(2,6)等点对没有相应的 Voronoi 多边形。值得注意的是，某些点并不在其相应 Voronoi 多边形内，例如，点 6 在点对(4,6)相应的 Voronoi 多边形内，但不在(5,6)、(6,8)、(6,7)等由点 6 组成点对的相应 Voronoi 多边形内，这一点显然与一阶 Voronoi 图不同。二阶 Voronoi 图与一阶 Voronoi 图的相同之处是，组成 Voronoi 图的 Voronoi 多边形均为凸多边形。我们仍采用一阶 Voronoi 顶点和 Voronoi 边的定义方式来定义二阶 Voronoi 顶点和 Voronoi 边。在图 4-42 中，Voronoi 边两侧的数字表示该边为其数字所标记点的垂直平分线，称为边权，比如，Voronoi 边$\frac{5}{4}$是点 4 与点 5 的垂直平分线，图中有些边未标数字。

$|S|=n, k=2$ 时，组成点对的数目为$\frac{n(n-1)}{2}$，即二阶 Voronoi 图至多由$\frac{n(n-1)}{2}$个 Voronoi 多边形组成。如果计算点对(p_i, p_j)相应的 Voronoi 多边形需要线性时间，那么计算$\frac{n(n-1)}{2}$个 Voronoi 多边形，即计算二阶 Voronoi 图需要 $O(n^3)$时间，减少所要产生的

Voronoi 多边形的个数以及计算每个 Voronoi 多边形的时间是降低复杂性阶的途径之一。利用反演几何的基本概念可以设计出构造高阶 Voronoi 图的算法。下面介绍分治法构造二阶 Voronoi 图，如图 4-43 所示。

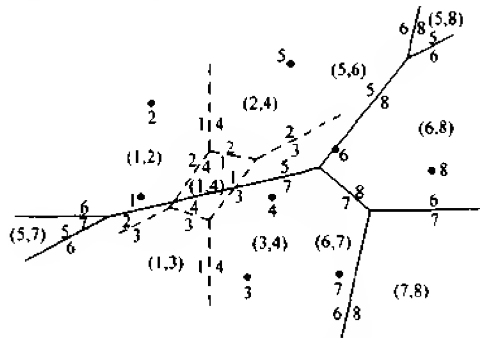


图 4-43　分治法构造二阶 Voronoi 图

$Z_{4\text{-}3}$算法（构造二阶 Voronoi 图的分治算法）

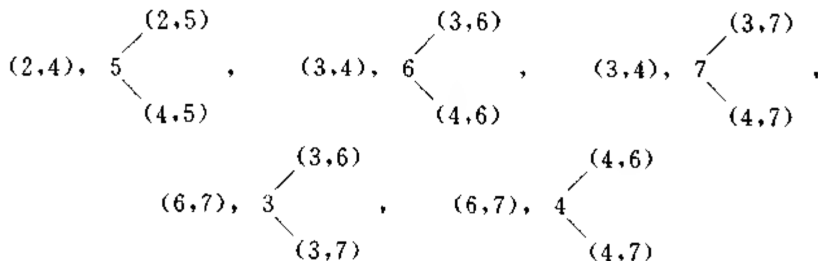
步 1　按点的 x 坐标分类 S 中的点。

步 2　分割 S 成若干子集 $S_k(k=1,2,\cdots,m)$，使得 $|S_k|\leqslant 4$。

步 3　按定义构造 $\mathrm{Vor}_2(S_k)$。

步 4　递归合并 $\mathrm{Vor}_2(S_{2i+1})$ 与 $\mathrm{Vor}_2(S_{2i+2})$，$i=0,1,\cdots,\left\lfloor\frac{n-1}{2}\right\rfloor$，直至构造出 $\mathrm{Vor}_2(S)$。

为了合并 $\mathrm{Vor}_2(S_{2i+1})$ 与 $\mathrm{Vor}_2(S_{2i+2})$，首先要判断子点集 S_{2i+1} 中的点有哪些点落入 $\mathrm{Vor}_2(S_{2i+2})$ 的无界 Voronoi 多边形内（其边不与 $\mathrm{Vor}_2(S_{2i+1})$ 的边相交），例如，图 4-43 中，$S_1=\{1,2,3,4\}$，其中点 3 与 4 落入 $\mathrm{Vor}_2(S_2)$ 中点对(6,7)相应的 Voronoi 多边形内（简称 3 与 4 落入(6,7)域中）。然后，计算新增加的点对及点对相应的 Voronoi 多边形，并修改已有点对相应 Voronoi 多边形的边界，例如图 4-43 中，点 3 落入(6,7)域内，便可能增加新点对(3,6)与(3,7)。再按定义计算这些新点对相应的 Voronoi 多边形的边界，计算结果(3,6)相应的 Voronoi 多边形为空，而(3,7)相应的 Voronoi 多边形非空。同样计算点 4 落入(6,7)域内所产生的(4,6)，(4,7)新域的边界。对于落入(2,4)域中的点 5，(3,4)域中的点 6 与点 7 同样处理。这样便得到图 4-42 所示的二阶 Voronoi 图。采用这种方法进行合并，不可能产生的点对首先被排除了，从而节省了计算时间，例如图 4-43 中可产生如下新点对：



删除重复出现的点对之后，只要计算(2,5)，(4,5)，(3,6)，(4,6)，(3,7)，(4,7)等6组点对相应Voronoi多边形的边界。计算结果显示(3,6)域为空，合并后的二阶Voronoi图增加了5个Voronoi多边形，另外要修改某些原Voronoi多边形的边界。

为了分析$Z_{4\text{-}8}$算法的时间复杂性，步4的递归合并是关键，只要估计出二阶Voronoi图中无界Voronoi多边形的数目，便可限界新增点对数目，从而求出合并步的复杂性。

周培德于最近设计了构造平面点集二阶Voronoi图的另一个算法，下面先介绍相关的概念与性质，然后描述算法。

给定三角形abc，其三条边的垂直平分线交于d，从d出发垂直平分线的延长线称为该垂直平分线的补线。在图4-44中，用虚线表示垂直平分线的补线。

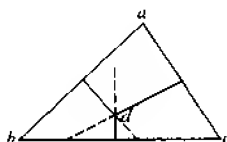


图4-44　垂直平分线的补线(虚线表示)

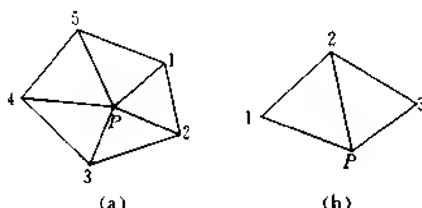


图4-45　与点p关联的三角形

如果多个三角形均以点p为公共顶点，并且这些三角形之间两两均有公共的边，它们围成一个圈(见图4-45(a))或者没有围成圈(见图4-45(b))，则称这些三角形与点p关联。

定理4-14　给定任意凸四边形$abcd$，若对角线$|ac|>$对角线$|bd|$，则$\overline{ac}$边的两端点对(a,c)不存在$Vor_2(a,c)$。

证明　引对角线$\overline{bd}$，$\overline{bd}$分割四边形$abcd$成两个三角形abd和bcd，分别作该两个三角形3条边的垂直平分线，这些垂直平分线的补线围成一个二阶Voronoi多边形。由补线的权值可以确定该多边形是点对(b,d)的二阶Voronoi多边形，而点对(a,c)不存在$Vor_2(a,c)$。如果引对角线$\overline{ac}$，则也得到同样的结论。

定理4-15　任意凸多边形$\{p_1,p_2,\cdots,p_n\}$，其直径$\overline{p_ip_j}$的两端点对(p_i,p_j)不存在$Vor_2(p_i,p_j)$。

证明　将凸多边形$\{p_1,p_2,\cdots,p_n\}$划分成若干个凸四边形和三角形，设$p_1p_2p_3p_4$是其中一个凸四边形，由定理4-14，该四边形的较长对角线(比如$\overline{p_1p_3}$)的两端点对不存在$Vor_2(p_1,p_3)$。然后增加相邻的凸四边形或三角形，得到凸六边形或凸五边形，如图4-46所示。图4-46中$p_2p_3p_6p_5$是新增加的相邻的凸四边形，该凸四边形的四个顶点不存在$Vor_2(p_2,p_6)$。两个四边形组成的凸六边形不存在$Vor_2(p_1,p_6)$，其中$\overline{p_1p_6}$是凸六边形的直径。如此不断增加顶点，直至n个顶点都被考查过，而且每次增加顶点之后，均不存在直径的两端点对所对应的二阶Voronoi多边形，因此定理的结论成立。

定理4-16　任意凸六边形至多有3个有界的二阶Voronoi多边形。

证明　将凸六边形用较短对角线划分成两个四边形，如图4-47所示。每个四边形只

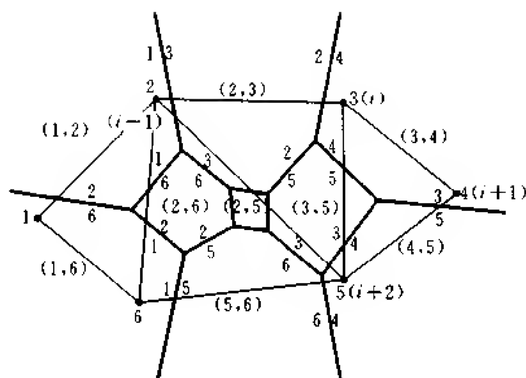


图 4-46　6 个点的二阶 Voronoi 图

有一个有界的二阶 Voronoi 多边形，此外分割线可能产生一个有界的二阶 Voronoi 多边形，但不会产生两个有界的二阶 Voronoi 多边形，因此结论成立。

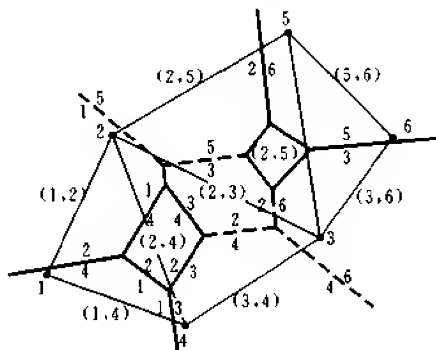


图 4-47　定理 4-16 证明的示意图

由定理 4-16 或者定理 4-15 可以推得，凸六边形中 9 条对角线至多有 3 条对角线，而且是最短的 3 条对角线的两端点对存在有界的二阶 Voronoi 多边形。对于任意凸多边形并且其内部包含有 S 中点，比如图 4-48 中的凸多边形内包含 3 个 S 中的点。将此凸多边形划分成 3 个凸四边形和 3 个三角形，每个四边形的较短对角线的两端点对对应一个有界的二阶 Voronoi 多边形，相邻四边形之间及相邻的四边形与三角形之间的公共边的两端点对可能产生一个有界的二阶 Voronoi 多边形。因此，图 4-48 中至多有 3+8=11 个有界的二阶 Voronoi 多边形，至少有 3+2=5 个有界的二阶 Voronoi 多边形，其中 2 是三个凸四边形之间两条公共边的端点对所对应的有界二阶 Voronoi 多边形的个数。这就减少了所要考虑的点对，即凸六边形中有 6 条对角线的端点对不存在对应的二阶 Voronoi 多边形。

一般说来，设凸多边形被划分为 k 个凸四边形和 l 个三角形，k 个凸四边形之间有 $k-1$ 条公共边，凸四边形与三角形之间有 $2l$ 条公共边($k=l=1$ 除外)，则有界的二阶

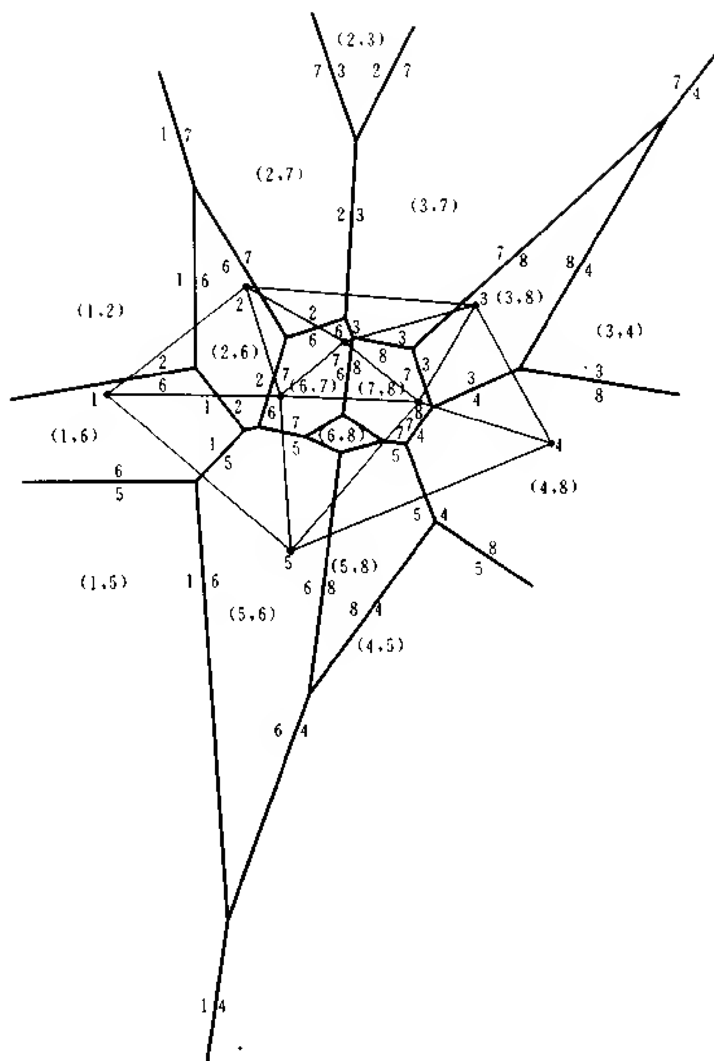


图 4-48　凸五边形内含有 3 个 S 中点

Voronoi 多边形的个数至多为 $k+(k-1)+2l=2(k+l)-1$，无界的二阶 Voronoi 多边形的个数大于或者等于凸多边形边的数目，如图 4-49 所示。当 S 中点都是凸壳顶点时，无界的二阶 Voronoi 多边形的个数与凸多边形边的数目相等，有界的二阶 Voronoi 多边形的个数等于最小权三角剖分边（凸多边形的边除外）的数目，如图 4-47 所示，而且边的端点对恰好与二阶 Voronoi 多边形的标记吻合。依据上述思想可以设计构造二阶 Voronoi 图的算法。

$Z_{4.9}$算法（构造平面点集二阶 Voronoi 图的算法）

输入　平面点集 $S=\{1,2,\cdots,n\}$及 n 个点的坐标

输出　二阶 Voronoi 多边形的集合即二阶 Voronoi 图

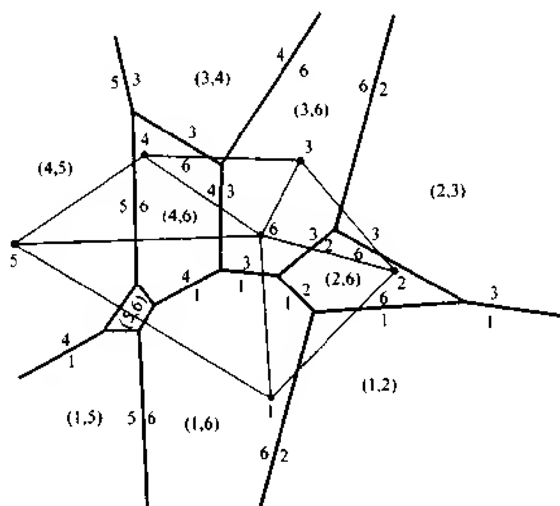


图 4-49　二阶 Voronoi 多边形个数达到限界的例子

步 1　计算点集 S 的凸壳，设凸壳为 CH。

步 2　对 CH 内点集 S（包含凸壳顶点）进行三角剖分，设三角剖分为 T（不包含凸壳边界）。

步 3　如果边 $e=(i,i+1)$ 是凸壳边界，比如图 4-46 中边 $e=(3,4)$，则计算边 e 关联的三角形（图 4-46 中三角形 345）3 条垂直平分线的补线，以及与 i 关联的三角形（图4-46中三角形 352）组成凸多边形（图 4-46 中凸四边形 3452）对角线（图4-46中对角线 42）的垂直平分线，它们组成该点对对应的 $\mathrm{Vor}_2(i,i+1)$。

步 4　如果边 $e\in T$，则按下述步骤计算边 e 所对应的二阶 Voronoi 多边形。

步 4-1　设以 $e=(i,j)$ 为公共边的两个三角形分别为 kij 和 ijl。

步 4-2　分别计算 kij 和 ijl 三边 kj、ki、ij 和 il、jl 的垂直平分线，它们分别交于 a、b。

步 4-3　计算垂直平分线的补线，补线分别交于 c、d，标记补线的边权。

步 4-4　以 a、b、c 和 d 为顶点的四边形 D_{ij} 是与边 (i,j) 对应的二阶 Voronoi 多边形，记为 $\mathrm{Vor}_2(i,j)$。

步 4-5　修改 D_{ij}。

步 4-5-1　从与点 j 关联的三角形中选取有公共边的三角形对，设顶点分别为 kju 和 jlv。

步 4-5-2　计算 $\overline{iu}$、$\overline{iv}$ 的垂直平分线，设为 f、f'。

步 4-5-3　检查 f、f' 是否切割多边形 D_{ij}。如果切割 D_{ij}，则修改 D_{ij}，并更换被考查的边，即 kj、lj 改为 uj、vj。重复步 4-5-1 至 4-5-3，直至多边形 D_{ij} 不被修改。否则，执行步 4-5-4。

步 4-5-4　对 ik、il 重复步 4-5-1 至 4-5-3，只是将 kj、lj 分别改为 ik、il，即从与点 i 关联的三角形中选取。执行步 4-5-3 时，如果 f、f' 均没有切割 D_{ij}，则 D_{ij} 为与边 (i,j) 对应的二阶 Voronoi 多边形。

步 5 输出结果。

定理 4-17 给定平面上 n 个点的点集 S，$Z_{4\text{-}9}$算法构造 S 的二阶 Voronoi 图的时间复杂性为 $O(n\log n)$。

证明 如果执行 $Z_{4\text{-}9}$算法中步 4-5 修改 D_{ij}时，处理与点 i、j 关联的每个三角形所得到的 f、f' 均切割 D_{ij}，那么多边形 D_{ij}的边界所含边的数目等于与 i、j 关联的三角形数目之和；否则小于与 i、j 关联的三角形数目之和。众所周知，平面点集 S 的三角剖分至多有 $3n-6$ 条边，这就是说点集 S 至多有 $3n-6$ 个二阶 Voronoi 多边形。与点集 S 中每个点关联的三角形的平均数不超过

$$2 \times \left\lceil \frac{3n-6}{n} \right\rceil = 6$$

因此，与 i，j 关联的三角形的平均数之和不超过 $6+6=12$，这表明多边形 D_{ij}的边界所含边的平均数不超过 12。所以点集 S 的二阶 Voronoi 图的边的总数至多为 $12\times(3n-6)=O(n)$。因此，如果假设计算一条二阶 Voronoi 边耗费 1 个单位时间，则 $Z_{4\text{-}9}$算法的步 3 和步 4 需要的时间为 $O(n)$。

另外，算法的步 1 和步 2 分别需要 $O(n\log n)$时间。因此 $Z_{4\text{-}9}$算法总共需要的时间是 $O(n\log n)$。

由二阶 Voronoi 图中多边形的标记可以产生出最小权三角剖分。例如图 4-47 中，最初给出的四边形是 1234，如果取对角线 13，则得到两个三角形 123 和 134，经 $Z_{4\text{-}9}$算法中步 4 处理后，得到有界多边形(2,4)，该标记产生四边形 1234 的最小权三角剖分，即连接顶点 2 与 4。这说明算法中步 2 得到的三角剖分不是最小权三角剖分时，经步 3 和步 4 处理之后可以得到点集 S 的最小权三角剖分。另外，由二阶 Voronoi 图的定义也可以说明这一点。S 中某些点对不存在二阶 Voronoi 多边形，表示该点对的连线不是最小权三角剖分的边。总之，平面点集的最小权三角剖分问题可以在多项式时间内求解，因而是 P 类问题。

4.4.9 几何数据压缩

几何数据压缩是指对描述场景的模型数据进行压缩，以便于模型数据的存储和传输，在分布式虚拟现实、协同应用、多用户视频游戏、模型数据在计算机内部的存储和传输等方面有着很重要的意义。也就是说，在上述各方面，必须进行数据压缩才能适应现有的网络传输技术，特别是在进行实时处理时更需要压缩技术的支持。

描述三维物体时，通常采用多边形网格来表示模型。而多边形网格表示的模型一般由三部分数据组成：几何数据，即多边形顶点的位置坐标；连接关系，即每个面的顶点及面与面之间的连接关系；特性，即各顶点的颜色、表面法向和纹理坐标等。数据压缩就是针对上述三部分数据的压缩(这里仅讨论几何数据和连接关系的数据压缩)。当前使用最多的是三角形网格组成的模型。

1995 年，Michael Deering 在 SIGGRAPH 上发表了一篇题为 Geometry Compression(几何压缩)的论文，受到人们的关注。Michael Deering 提出的方法是基于通用三角形网格的几何数据压缩，其压缩比为 6/1 至 10/1。之后，Stefan Gumhold 于 1998 年提出三角

形网格连接关系实时压缩算法，其压缩比提高到 8/1 至 12/1。与此同时，Gabriel Taubin 的基于拓扑手术的几何数据压缩算法将压缩比提高到 50/1。上述三种算法都是处理单分辨率模型的几何数据压缩问题。下面先简要介绍 Gabriel Taubin 算法，然后叙述周培德提出的算法并进行比较。

三角形网格所占用的存储空间依赖于其表述方法。最简单的方法是每个三角形独立地由其 3 个顶点的 9 个坐标来表述，这需要 36 个字节才描述一个三角形。这种方法使每个顶点约被描述 6 次，浪费了存储空间。如果将顶点和三角形分开表示，每个顶点用 3 个坐标确定，而每个三角形用三角形描述表来表示，其中每个表项只包含该三角形的 3 个顶点序号，那么这将节省大量存储空间。另外，引入三角形条带的方法可以进一步减少不必要的顶点传输，即两个相邻的三角形共用一条边，这也就是说，从第二个三角形开始，一个新顶点的加入和其前一个三角形的一条边即可构成一个新的三角形。这样，每个顶点最多必须传送两次。

Gabriel Taubin 提出了顶点生成树和三角形生成树概念，从构造顶点生成树出发，沿顶点树切割得到三角形树，对树的编码即得到连接关系，这就是 Gabriel Taubin 算法的基本思想。其步骤如下。

步 1 构造顶点生成树 在三角形网格中选一点作为顶点树的根节点，包含该节点的单连通域的边界是第一层边界。依此类推，三角形网格划分成嵌套的环。在这些环中，内层顶点数大约是相邻外层顶点数的一半，对应于根节点的点，其度数约为 6。将这些环转变成螺旋状连线之后可以构造顶点树和三角形树，这只要在第 k 层边界边中选择一顶点和第 $k+1$ 层边界边中一顶点连接成跨越边，那么各层边界边便连接成一螺旋线，如图 4-50 所示。

步 2 顶点树编码

步 3 顶点坐标的压缩

步 4 三角形树编码

步 5 计算和压缩行进模式

该算法对网格中三角形连接关系的压缩是无损的，而对顶点坐标和属性坐标的压缩是有损的。

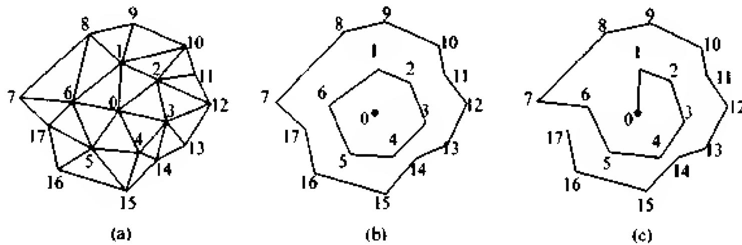


图 4-50 构造顶点生成树

Gabriel Taubin 算法需要在多边形内部增加一定数量的点，然后进行三角剖分，并将这些三角形划分成嵌套的三角形环。能否省去增加点的工作，而直接对多边形内部进行三

角剖分并建立三角形顶点之间的关系，使多边形每个顶点只存储一次，传输一次？回答是肯定的。下面的算法正是基于这一思想设计的。

$Z_{4\text{-}10}$算法（几何数据压缩的算法）

步 1 确定多边形 L 各顶点的凸、凹性。

步 2 依次连接凹点，构成一新的多边形 L_1。$L_1 \subset L$，如若不然，则修改 L_1（增加 L 的若干凸点），使 $L_1 \subset L$。

步 3 对 L_1 重复步 1。设 p_i、p_{i+1}、p_{i+2}是 L_1 的凹点（L_1 无凹点、或有 1 个、2 个凹点时，类似处理）。

步 4 选凹点 p_{i+1}及 L_1 的另一顶点 p_j，使得 p_{i+1}与 p_j 之间的顶点数目和 p_j 与 p_{i+1}之间的顶点数目大致相等。

步 5 连接 p_{i+1}与 p_j 成一线段，作为顶点树的根结点（含两个编号，即 p_{i+1}与 p_j）。

步 6 位于$\overrightarrow{p_j p_{i+1}}$左侧（右侧）并与 p_j、p_{i+1}相邻的 L_1 的顶点作为根结点的左子结点（右子结点），它们组成凸四边形或三角形。

步 7 顶点树中其他顶点按下列规则生成：

步 7-1 i 级、$i+1$ 级结点（且满足父子关系）均包含两个顶点编号时，$i+2$ 级的子结点有 3 个，分别表示 3 个三角形，即 i 级、$i+1$ 级结点中各取一顶点编号（同时左或右）与 $i+2$级左或右子结点组成一个三角形，或者 $i+1$ 级一结点中两顶点编号与 $i+2$ 级居中结点组成一个三角形。

步 7-2 i 级结点包含两个顶点的编号，而 $i+1$ 级、$i+2$ 级结点均包含一个顶点编号时，位于 i 级结点中左（右）顶点编号、与 $i+1$ 级居中顶点编号、$i+2$ 级左（右）子顶点编号组成一个三角形。

步 7-3 其他类似规则，不一一列举。

L 的顶点树产生之后，只需存储该顶点树，并按广度优先顺序传输。L 的每个顶点只需存储一次，传输一次，不需要重复存储和传输，也不需要增加点，从而减少了存储量和计算量。$Z_{4\text{-}10}$算法进一步提高了压缩比。由于多边形 L 的顶点连接关系蕴含于顶点树之中，所以对三角形连接关系的压缩是无损的。此外，$Z_{4\text{-}10}$算法又不需要对顶点坐标进行四则运算，因此对顶点坐标的压缩也是无损的。总之，从多项指标来分析，$Z_{4\text{-}10}$算法优于 Gabriel Taubin 算法。另外，$Z_{4\text{-}10}$算法也适用于三维情况。

第5章 交 与 并

实际应用中常常碰到交问题，比如，消除隐藏线的问题是构造两个多边形的交，如果没有构造多边形交的最优算法，那么就不可能有效地消除隐藏线；再比如，在模式识别、导线和元件布局、线性规划等领域或问题中都会碰到判定几何体是否相交或确定它们的交的问题。并问题也常常出现于工程中，比如，确定具有多个多边形（平面图）组合的建筑群体所占的平面范围。为了有效地解决这些问题，必须设计出求两个几何对象的交或者并的有效算法。

由于两个几何对象相交，仅当一个几何对象至少包含另一个几何对象的一点时才成立，所以交算法涉及到包含的测试。本章介绍线段交的算法、多边形交的算法、多边形并的算法以及多面体交的算法。

5.1 线段交的算法

给定平面上 n 条线段，确定其中任意两条线段是否相交；如果相交，则求出所有的交点。平面多边形可以看成是平面上线段的集合，判定两个多边形是否相交的问题可以变换为两个线段集合中的线段是否相交的问题，而且这种变换在多项式时间内可以完成，因此有

$$测试多边形相交问题 \propto 测试线段相交问题$$

即测试多边形相交的问题可以在多项式时间内转换为线段相交问题。另外，多边形是否为简单多边形的测试问题也可以在多项式时间内变换到测试线段相交问题。

先考虑一维情况。在 x 轴上给定 n 个区间 $[x_1, x_2], [x_3, x_4], \cdots, [x_{2n-1}, x_{2n}]$，确定其中任意两个区间是否重叠。下述算法求解该问题。

判定 n 个区间是否重叠的算法

步 1 分类 n 个区间的端点，表示为 $x_1 < x_2 < \cdots < x_{2i-1} < x_{2i} < \cdots < x_{2n-1} < x_{2n}$

步 2 **if** $(x_2 < x_3) \land (x_4 < x_5) \land \cdots \land (x_{2n-2} < x_{2n-1})$

　　then n 个区间不重叠

　else if $x_{2i} > x_{2i+1}$

　　　then 第 i 个区间与第 $i+1$ 个区间重叠

该算法步 1 分类需要 $O(n\log n)$ 时间，步 2 耗费线性时间便可检查 $n-1$ 个不等式 $x_{2k} < x_{2k+1}$ $(k=1,2,\cdots,n-1)$ 是否成立，故算法的时间复杂性为 $O(n\log n)$。可以证明，确定 n 个区间是否不相交的问题的下界是 $O(n\log n)$。因此该算法是解决区间重叠问题的最优算法。

对于二维情况，假设平面上给定 n 条线段，利用对每两条线段检查是否相交的方法，可以确定 n 条线段中哪些线段相交。这种时间复杂性为 $O(n^2)$ 的算法是最简单的，但当 n

很大时，这种方法将耗费相当多的时间，因此需要寻找新的算法。

我们先建立平面上线段之间的次序关系。

给定平面上两条不相交的线段 s_1 和 s_2，如果存在一条直线通过 x 轴上的点 x_0，且垂直于 x 轴，此直线与 s_1、s_2 交于 y_1、y_2 且 $y_1>y_2$，则称 s_1 与 s_2 是可以比较的，记作 $s_1>_{x_0}s_2$，如图 5-1 所示。反之称 s_1 与 s_2 是不可比较的。

在图 5-2 中，有 s_1，s_2，s_3，s_4 等 4 条线段，其中 s_1，s_2，s_4 关于 x_0，x_1 是可比较的。且有关系式：

$$s_2>_{x_0}s_4;\quad s_2>_{x_1}s_4$$

$$s_1>_{x_1}s_2;\quad s_1>_{x_1}s_4$$

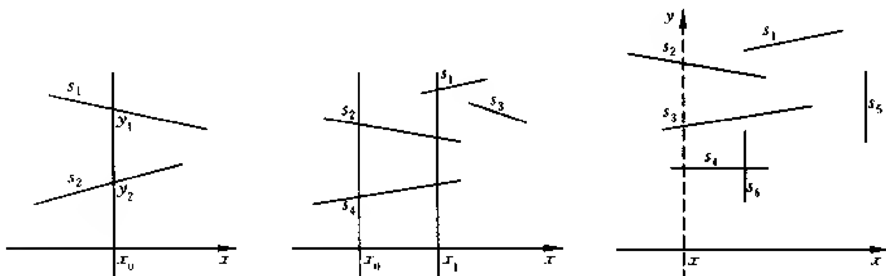


图 5-1 可比较的两条线段 图 5-2 s_3 与 s_1、s_2、s_4 不可比较 图 5-3 y 轴进行水平扫描

图 5-2 中，s_3 与 s_1，s_2，s_4 是不可比较的。这里不可比较的原因是没有与 x 轴垂直的直线与 s_3 相交。显然，当两线段不可比较时，此两线段不可能相交，即两线段相交的必要条件是有某个 x，使关系式 $s_1>_x s_2$ 成立。因此当考虑线段相交问题时，只要分析那些可以比较的线段而不必考虑不能比较的线段。

在图 5-2 中，只要作下述改进，s_3 与 s_1，s_2，s_4 便可以进行比较。

想象有一条垂直于 x 轴的直线(即 y 轴，又称扫描线)，自左向右扫过所有的线段(见图 5-3)。各线段在 y 轴上的投影不是点就是区间。随着 y 轴由左向右移动，这些投影点和投影区间出现后又消失了。如果一条水平线段的投影点落在另一条垂直线段的投影区间中间，则此两线段相交，如图 5-3 中的 s_4 和 s_6。非水平与非垂直线段的相交问题，不能这样来判定。图 5-3 中的 s_1，s_2，s_3，s_4，s_5，s_6 经过对 x 坐标的排序，次序是("$<s_1$"表示线段 s_1 的始点，"$>s_1$"表示 s_1 的终点)：

$$<s_2<s_3<s_4<s_1<s_6s_6>>s_2>s_4>s_3>s_1<s_5s_5>$$

利用上述改进的方法，图 5-2 中的线段 s_3 与 s_1 有关系：$s_1>_x s_3$。

因此，用平移的 y 轴可以对所有有关线段建立全序关系"$>_x$"。该次序关系仅以三种方式变化：

(1) 遇见线段 s 的左端点，将 s 加入该次序关系。

(2) 遇见线段 s 的右端点，此时从次序关系中删去 s，因为 s 不再与其他线段可比较。

(3) 达到两条线段 s_1 与 s_2 的交点 p，在次序关系中于 p 处 s_1 和 s_2 交换位置。

如果线段 s_1 和 s_2 相交，当移动的 y 轴接近线段 s_1 与 s_2 交点的 x 坐标时，s_1 与 s_2 成为

可比较的，因而 s_1 与 s_2 一定在这全序关系中成为相邻的两条线段。不必对每条线段检查是否与其他所有线段相交，只要检查在全序关系中相邻线段是否相交即可。利用分类算法可以实现 y 轴的移动，从而建立全序关系。

平面扫描方法用了两个基本的数据结构：扫描线状态和事件点进度表。扫描线状态是关系 $>_x$ 的一种描述，即它是线段的一个序列。在平面扫描中关系 $>_x$ 在 x 坐标的有限集内变化，显然，实现扫描线状态的数据结构 T 一定支持下列的操作：

(1) INSERT(s,T)，表示将线段 s 插入由数据结构 T 保持的全序关系中。

(2) DELETE(s,T)，表示从 T 中删去线段 s。

(3) ABOVE(s,T)，返回 T 中 s 上面的一个相邻线段。

(4) BELOW(s,T)，返回 T 中 s 下面的一个相邻线段。

支持上述操作的数据结构 T 选择词典，为了有效地实现这种数据结构，我们采用均衡二叉树。这样，实现 INSERT、DELETE 操作的时间与存储在数据结构中的元素个数的对数成比例，即执行插入、删去操作耗费对数时间，而 ABOVE、BELOW 操作只需要常数时间。

事件点进度表包含线段的左、右端点及线段交点的 x 坐标。当扫描线扫描平面时，一定要保持关系 $>_x$，这种关系只是在扫描线遇到左、右端点及交点的 x 坐标时才发生变更。当预先给定所有线段的左、右端点时，由平面扫描找到一个交点 p，这就动态地产生一个事件，产生一个事件和处理该事件在时间上常常有差距。这就是说，算法对刚产生的事件（找到的交点）可能不马上处理（交换两线段的位置等），而是先处理其他事件，比如插入某些线段。为了找到所有的交点，实现事件点进度表的数据结构 E 将支持下列操作：

(1) MIN(E)，确定 E 中的最小元素，并删去它。

(2) INSERT(x,E)，将 x 插入到由 E 保持的全序中去。

(3) MEMBER(x,E)，确定 x 是否是 E 的一个成员。

数据结构 E 选择优先队列，并且仍采用均衡二叉树实现这种数据结构，因此实现上述三种操作耗费对数时间。

扫描线扫描平面时，在事件点处修改数据结构 T，T 中凡是变成相邻的线段对便被检查是否相交。如果查出一个交点，则打印形成该交点的两条线段，而且把交点的 x 坐标插入到事件点进度表 E 中去。下面是 Bentley-Ottmann 提出的判定线段相交的算法。

begin

1. 按 x、y 词典式地分类 $2n$ 个端点，并把它们放入优先队列 E。
2. $A \leftarrow \varnothing$
3. **while** $E \neq \varnothing$ **do**

    begin

4. $p \leftarrow$ MIN(E);
5. **if** p 是左端点 **then**

    begin

6. $s \leftarrow p$ 是端点的线段;
7. INSERT(s,T);

```
8.      s1←ABOVE(s,T);
9.      s2←BELOW(s,T);
10.     if  s1 和 s 相交  then  A←(s1,s);
11.     if  s2 和 s 相交  then  A←(s,s2)
    end
12. else  if  p 是右端点  then
    begin
13.     s←p 是端点的线段;
        s1←ABOVE(s,T);
14.     s2←BELOW(s,T);
15.     if  s1 和 s2 交于 p 的右边  then  A←(s1,s2);
16.     DELETE(s,T)
    end
17.   else /p 是一个交点/
      begin
18.       (s1,s2)←交点是 p 的线段
          /p 的左边有 s1=ABOVE(s2)/
19.       s3←ABOVE(s1,T);
20.       s4←BELOW(s2,T);
21.       if  s3 和 s2 相交  then  A←(s3,s2);
22.       if  s1 和 s4 相交  then  A←(s1,s4);
23.       在 T 中交换 s1 和 s2
      end
      /处理查出的交点/
24. while  A≠∅  do
    begin
25.     (s,s')←A;
26.     x←s 和 s'交点的横坐标;
27.     if  MEMBER(x,E)=FALSE  then
        begin
28.       输出(s,s');
29.       INSERT(x,E)
        end
    end
  end
end
```

该算法仅在相邻线段中寻找交点，且所有相邻的线段至少被检查一次，所以算法不会遗漏交点。算法第 1 行耗费 $O(n\log n)$时间完成分类。6～11，13～16 和 18～23 行是互不

相交的程序段，它们分别用时间 $O(\log n)$，这是因为 T 上的每个操作在 $O(\log n)$ 时间内均能完成，而且判定两条线段是否相交只要常数时间。每次执行 while 循环时，只执行上述三个程序段中的一段及 24～29 行，后者的执行时间为 $O(\log n)$。若交点数是 k，则 while 循环要执行 $(2n+k)$ 次，每次 while 循环耗费 $O(\log n)$ 时间，因此算法的时间复杂性为 $O((2n+k)\log n)$。由于平面上 n 条线段的交点数可以达到 $\frac{n(n-1)}{2}=O(n^2)$，所以 $k \leqslant O(n^2)$，因而算法的时间复杂性为 $O(n^2\log n)$。

确定平面上 n 条线段所有交点这一问题的时间下界是 $\Omega(k+n\log n)$，因此上述算法不是最优的。1988 年，Chazelle 和 Edelsbrunner 给出了一个时间复杂性为 $\theta(k+n\log n)$ 的算法，是解决该问题的最优算法。但是后者的空间复杂性为 $O(n+k)$。高于前者的空间复杂性 $O(n)$。能否找到时间和空间都达到下界的最优算法呢？这是一个有待解决的问题。

如果只是需要寻找 n 条线段中的一个交点而不是所有交点，那么问题就简单了。此时 T 不变而 E 取为数组，它存储 n 条线段的 $2n$ 个端点。算法描述如下：

```
begin
  按 x、y 坐标词典式分类 2n 个端点，并将它们存入数组 POINT[1··2n]；
  for  i=1  to  2n  do
    begin
      p←POINT[i]；
      s←p 为端点的线段；
      if  p 是左端点  then
        begin
          INSERT(s,T)；
          s1←ABOVE(s,T)；
          s2←BELOW(s,T)；
          if  s1 和 s 相交  then  输出(s1,s)；
          if  s2 和 s 相交  then  输出(s2,s)
        end
      else  /p 是 s 的右端点/
        begin
          s1←ABOVE(s,T)；
          s2←BELOW(s,T)；
          if  s1 和 s2 相交  then  输出(s1,s2)；
          DELETE(s,T)
        end
    end
end
```

该算法求得最左边的一个交点 a，这是因为扫描线从左边接近 a 时，交于 a 的两条线段成为相邻的。算法中 **for** 循环执行 $O(n)$ 次，每次耗费 $O(\log n)$ 时间进行相交测试，因此

算法的时间复杂性为 $O(n\log n)$，即用 $O(n\log n)$ 时间可以确定平面中 n 条线段的任何两条是否相交，而且这是最优的。

由于判定两个多边形是否相交以及判定多边形是否是简单多边形等问题，可以多项式变换到判定平面线段相交问题，而判定平面线段相交问题在时间 $O(n\log n)$ 内可以求解，所以判定两个多边形是否相交和判定多边形是否为简单多边形问题可以在时间 $O(n\log n)$ 内求解。

上述两个算法中，只是告诉判定哪两条线段相交，而没有说明如何判定两条线段相交，下面将通过引入两个函数 Same 与 Intersect 计算这个问题。

设给定平面上两条线段 s_1 与 s_2，s_1 的两个端点为 p_1 和 p_2，如果 p_1、p_2 分别在 s_2 的两侧，同时 s_2 的两个端点分别在 s_1 的两侧，则 s_1 与 s_2 相交。否则不相交。

用下述形式表示点和直线段：

```
type  point:=record x,y:integer
type  line:=record p1,p2:point
```

根据上面的想法，要引入两个函数，一个是判断线段 s_1 的两个端点是否在 s_2 的同一侧，即 same(s:line;p_1,p_2:point): boolean，此函数当 p_1、p_2 位于 s 的同一侧时，返回值"true"，否则返回值"false"；另一个是判断线段 s_1 与 s_2 是否相交，即 intersect(s_1,s_2:line): boolean，当线段 s_1 和 s_2 相交时，返回值"true"，否则返回值"false"。引入这两个函数之后，就不必求 s_1 与 s_2 的交点，也能判断 s_1 与 s_2 是否相交。

```
function  same(s: line;p1,p2:point):boolean
Var  dx,dy,dx1,dx2,dy1,dy2:real
begin
  dx←s.p2.x—s.p1.x
  dy←s.p2.y—s.p1.y
  dx1←p1.x—s.p1.x
  dy1←p1.y—s.p1.y
  dx2←p2.x—s.p2.x
  dy2←p2.y—s.p2.y
  same←(dx*dy1-dy*dx1)*(dx*dy2-dy*dx2)>0
end
```

直线 s 的方程表示为

$$\Delta(x,y) = dx \cdot (y - s.p_i.y) - dy \cdot (x - s.p_i.x) = 0,\quad i = 1,2.$$

凡在 s 上的点便使 $\Delta(x,y)=0$，而 s 两侧的点必有 $\Delta(x,y)\neq 0$（一侧使 $\Delta(x,y)>0$，另一侧使 $\Delta(x,y)<0$）。所以如果 p_1、p_2 在 s 的同一侧，必然使 $\Delta(p_1.x,p_1.y)$ 和 $\Delta(p_2.x,p_2.y)$ 同时为正或负，即 p_1 与 p_2 在 S 的同侧的充分必要条件是 $\Delta(p_1.x,p_1.y)\cdot\Delta(p_2.x,p_2.y)>0$。同理，$p_1$ 与 p_2 在 s 的异侧的充分必要条件是 $\Delta(p_1.x,p_1.y)\cdot\Delta(p_2.x,p_2.y)<0$。

```
function  intersect(s1,s2: line):boolean
begin
  intersect←not same (s1,s2.p1,s2.p2)
```

```
        and not same(s2, s1. p1, s1. p2)
    end
```

如果两条线段的两个端点都在对方线段的两侧，则此两线段相交。

具体计算两条线段的交点时，先将线段表示为参数方程，然后求解参数方程。

设线段 s_1 的两个端点为 $a(a_0,a_1)$ 和 $b(b_0,b_1)$，a 点所表示的向量是原点至点 a 的向量，线段 s_1 用向量式表示为 $\vec{s_1}=\vec{b}-\vec{a}$，$\vec{s_1}$ 的方向即线段方向，如图 5-4 所示。线段 s_1 上的任意点表示为向量和 $s_1(t_1)=\vec{a}+t_1\vec{s_1}$，$t_1$ 是参数。当 $t_1=0$ 时，$s_1(t_1)=\vec{a}$，即 a 点；$t_1=1$ 时，$s_1(t_1)=\vec{b}$，即 b 点；$0\leqslant t_1\leqslant 1$，$s_1(t_1)$ 又表示线段 s_1 上的所有点。类似地，另一条线段 s_2（端点为 $c(c_0,c_1)$，$d(d_0,d_1)$）可表示为 $s_2(t_2)=\vec{c}+t_2\vec{s_2}$，$\vec{s_2}=\vec{d}-\vec{c}$，$1\leqslant t_2\leqslant 1$。为了计算 s_1 与 s_2 的交点，只要求解方程 $\vec{a}+t_1\vec{s_1}=\vec{c}+t_2\vec{s_2}$，解得

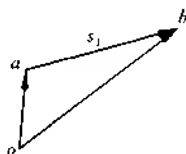


图 5-4　线段的向量表示

$$t_1=[a_0(d_1-c_1)+c_0(a_1-d_1)+d_0(c_1-a_1)]/D$$

$$t_2=-[a_0(c_1-b_1)+b_0(a_1-c_1)+c_0(b_1-a_1)]/D$$

$$D=a_0(d_1-c_1)+b_0(c_1-d_1)+d_0(b_1-a_1)+c_0(a_1-b_1)$$

5.2　多边形的交

本节介绍凸多边形的交、星形多边形交与一般简单多边形交的算法。

5.2.1　凸多边形交的算法

平面上给定两个凸多边形 $P=\{p_1,p_2,\cdots,p_n\}$ 与 $Q=\{q_1,q_2,\cdots,q_m\}$，其顶点按逆时针方向排列，确定它们的交，记为 $P\cap Q$。$P\cap Q=\{q|q\in P\wedge q\in Q\}$。显然，两个凸多边形的交是一个凸多边形。如果 P、Q 分别有 n，m 个顶点，那么 $P\cap Q$ 至多有 $n+m$ 个顶点，如图 5-5 所示。由图 5-5 可以看出，$P\cap Q$ 的边界由 P 边和 Q 边的交错序列组成，P 边与 Q 边的交点恰好是 $P\cap Q$ 的边界序列中 P 边与 Q 边的交错之处。



图 5-5　两个凸多边形的交

下面介绍几种求 $P\cap Q$ 的算法。

对 Q 的每条边检查 P 的所有边，判断它们是否相交，如果相交，则记录交点，然后输出 P、Q 边在交点处轮流出现的表，即 $P\cap Q$ 的边界。该算法描述如下。

确定 P∩Q 的穷举算法

输入　P 的边 $e_1,e_2,\cdots,e_n$ 及顶点序列 $p_1,p_2,\cdots,p_n$；Q 的边 $e_1',e_2',\cdots,e_n'$ 及顶点序列 $q_1,q_2,\cdots,q_m$。

输出　$P\cap Q$ 的边 $e_1'',e_2'',\cdots,e_l''$ 及顶点序列 $r_1,r_2,\cdots,r_l$。

```
    for  i=1  to  n  do
```

```
    for  j=1  to  m  do
      begin
      if  e_i 与 e'_j 相交
          then 求交点并记录交点
      end
  write  P∩Q 的边序列及顶点序列
```

if 语句耗费常数时间，双重循环的时间复杂性为 $O(mn)$，即该算法的时间复杂性。

构造 $P\cap Q$ 的另一种算法是，将 P,Q 顶点按 x 坐标分类，并通过各顶点作垂直于 x 轴的直线，如图 5-6 中虚线所示。在相邻两条虚线之间(即一个长条域内)，P、Q 各形成一个梯形(某些相邻虚线之间只有一个梯形或三角形)，在常数时间内可以计算这两个梯形的交(图 5-6 中阴影部分)。一般情况下，有 $n+m-1$ 个长条域，处理每个长条域用常数时间，如果不计分类时间，则算法的时间复杂性为 $O(n+m)$。若考虑分类时间，那么算法的时间复杂性为 $O((n+m)\log(n+m))$。

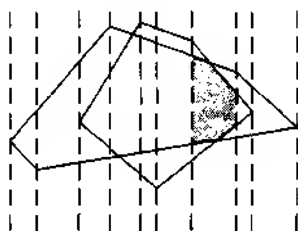


图 5-6　两个梯形的交组成 $P\cap Q$ 的一部分

梯形交组成 $P\cap Q$ 的算法

预处理　将 P、Q 顶点按 x 坐标分类。

步 1　过 p_i、q_j 作垂直于 x 轴的直线($i=\overline{1,n}$,$j=\overline{1,m}$)，形成数目不超过 $n+m-1$ 个长条域。

步 2　对每个长条域，计算两个梯形的各顶点(有的长条域内只有一个梯形或一个三角形)及两个梯形的交，交记为 W_i。

步 3　计算 $P\cap Q=\bigcup W_i$

第 3 种构造 $P\cap Q$ 的算法是，逐步沿 P 边和 Q 边行进，找出交点，然后确定 $P\cap Q$。

沿 P 边和 Q 边行进寻找 $P\cap Q$ 的算法

```
begin
  i←1, j←1, k←1
  repeat
    begin
      if  p_i p_{i+1} 与 q_j q_{j+1} 相交  then  记录交点
      增加 i 或增加 j(沿 P 边或 Q 边行进)
      k←k+1
    end
  until k=2(n+m)
  if  没有交点  then
    begin
      if  p_i∈Q  then  P⊆Q
      else  if  q_i∈P  then  Q⊆P
```

else $P \cap Q = \varnothing$

end

end

沿 P 边或 Q 边行进的方法如下：开始时，选任一条边，比如选 Q 边行进。然后，当 Q 的现时边$\overline{q_jq_{j+1}}$上已找到一个交点时，将改沿 P 边上行进；或当 P 的现时边$\overline{p_ip_{i+1}}$上已找到一个交点时，将改沿 Q 边上行进；或者 P 的现时边上的所有交点已被找到，而 Q 的现时边仍有交点未找到，则在 P 边上行进。如图 5-7 所示，图中$\overline{p_{i+1}p_{i+2}}$上两个交点已找到，而$\overline{q_{j+1}q_{j+2}}$上仍有交点未找到，则在 P 边上行进。repeat 语句循环 $2(n+m)$次，每次循环耗费常数时间，因此 repeat 循环耗费 $O(n+m)$时间，其他语句的复杂性不超过线性时间，因此算法的时间复杂性为 $O(n+m)$。

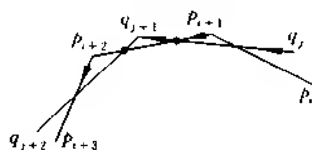


图 5-7 沿 P 边或 Q 边行进

下面介绍另外两个算法，它们均由周培德提出。

$Z_{5\text{-}1}$算法（计算多边形 $P=\{p_1, p_2, \cdots, p_n\}$与多边形 $Q=\{q_1, q_2, \cdots, q_m\}$的交）

步 1 求点 $p_1, p_2, \cdots, p_n$ 的 x 坐标的最小(大)值，设 $p_a(p_b)$的 x 坐标值最小(大)。求点 $q_1, q_2, \cdots, q_m$ 的 x 坐标的最小(大)值，设 $q_a(q_b)$的 x 坐标值最小(大)。

步 2 **if** $p_b < q_a \vee q_b < p_a$ **then** $P \cap Q = \varnothing$

else if $q_a < p_a < q_b$(p_a 在 Q 内)

then goto 步 3

步 3 从 p_a、q_a 出发，轮流检查与 p_a、q_a 关联的边及其后继边(分上、下两支)的端点对是否在对方边的两侧。

if 端点对在对方边的两侧

then 求该两条边的交点

else 对后继边同样处理，直到 p_b、q_b 被处理

步 4 从 p_a 出发沿 P 边行进，碰到交点改沿另一多边形边行进，直至回到 p_a，其路径即 $P \cap Q$ 的边界。

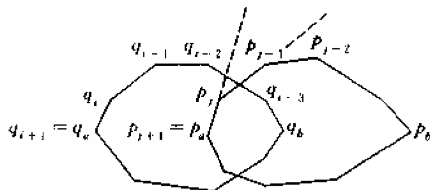


图 5-8 $Z_{5\text{-}1}$算法示意图

执行步 3 的具体方法如下：从 q_i 出发，判定 $q_i, q_{i-1}, \cdots$是否在$\overrightarrow{p_{j+1}p_j}$的右侧，如图 5-8 所示。在图 5-8 中，$q_{i-3}$与 q_{i-2}在$\overrightarrow{p_{j+1}p_j}$两侧。然后判定 $p_j, p_{j-1}, \cdots$是否在$\overrightarrow{q_{i-2}q_{i-3}}$的左侧，如图 5-8 所示。在图 5-8 中，$p_{j-1}$与 p_j 在$\overrightarrow{q_{i-2}q_{i-3}}$两侧。因此$\overline{p_{j-1}p_j}$与$\overline{q_{i-3}q_{i-2}}$相交，求出交点，这是第 1 个交点。再用沿 P 边和 Q 边行进的方法寻找所有其他交点，或者用判定一个多边形边的两个端点是否在另一多边形边的两侧的方法求出所有交点。

$Z_{5\text{-}1}$算法中的步 1 只需要线性时间，步 2 亦为线性时间。一条边上至多有两个交点，故需要检查常数次，由于这个原因，步 3 至多要检查 $C(n+m)$对边是否相交，其中 C 为常

数。检查两条边是否相交耗费常数时间，因此步 3 耗费时间为 $O(n+m)$。步 4 的时间不超过 $O(n+m)$，由于多边形边的顺序已经给定，该算法不需要将顶点分类，所以算法的时间复杂性为 $O(n+m)$。

另外，利用平面扫描方法也可以计算 $P\cap Q$。只要注意到多边形的边和顶点序列已经给定，而且每个顶点既是一条边的左端点，同时又是另一条边的右端点，就不难利用平面扫描方法求 $P\cap Q$。

利用 5.1 节中平面扫描判断线段集中哪些线段相交的方法求 $P\cap Q$ 时，作如下修改：由于扫描到一条线段右端点时必是另一条线段的左端点，因此只需在插入边时进行判断，而在删去边时不必进行判断。另外，任何时刻的扫描线至多碰到 4 条线段，所以问题比较容易求解。这样，求出所有交点后，再用 $Z_{5\text{-}1}$ 算法中的步 4 方法找 $P\cap Q$ 的边界。

$Z_{5\text{-}2}$算法（平面扫描方法计算 $P\cap Q$）

步 1　按 x 坐标分类 $n+m$ 个点，P 顶点放入 E_1，Q 顶点放入 E_2

步 2　**while**　$E_1\neq\varnothing\wedge E_2\neq\varnothing$　**do**

```
begin  p←MIN(E1∪E2)/剩余事件点集中 x 坐标最小值对应的点作为 p/
  if  p 是左端点  then
    begin  s←p 是左端点的线段；
      INSERT(s,T)；
      s1←ABOVE(s,T)；
      s2←BELOW(s,T)；
      if  s1 与 s 相交
        then  求交点，并记录交点；
      if  s2 与 s 相交
        then  求交点，并记录交点；
    end
  if  p 是右端点  then
      begin
        s←p 是右端点的线段；
        DELETE(s,T)，但暂保留 p 点(删去 s 的左端点)
      end
end
```

步 3　从点 Max (MIN(E_1),MIN(E_2))出发，用 $Z_{5\text{-}1}$算法中步 4 的方法求 $P\cap Q$ 的边及顶点序列并输出。

图 5-9 示出 $Z_{5\text{-}2}$算法的执行过程：扫描线首先碰到点 q_1，它是两条线段 s_1 与 s_2 的左端点，将此两条线段加入 T；s_1 在 s_2 上方，不相交；右移扫描线到 q_2，删去$\overline{q_1q_2}$，保留点 q_2；插入$\overline{q_2q_3}$；扫描线右移碰到 p_1，插入$\overline{p_1p_2}$，$\overline{q_2q_3}$在$\overline{p_1p_2}$上方，$\overline{p_1p_2}$与$\overline{q_2q_3}$相交，求出交点并记录交点；扫描线右移到 p_2，删去$\overline{p_1p_2}$，保留点 p_2；插入$\overline{p_2p_3}$，$\overline{p_2p_3}$与$\overline{q_2q_3}$成为新的相邻关系，经判断，它们相交，求出交点并记录交点。依此类推。

利用 $Z_{5\text{-}2}$算法计算 $P\cap Q$ 时，每条边只插入一次，但一条边可能与另一多边形的两条

边相交。交点数可以达到 $n+m$ 个，所以 $Z_{5\text{-}2}$ 算法的时间复杂性为 $O(n+m)$。如果考虑分类时间，则时间复杂性为 $O((n+m)\log(n+m))$。

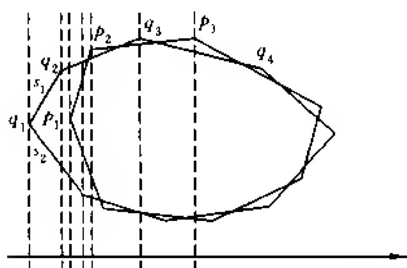


图 5-9　$Z_{5\text{-}2}$ 算法示意图

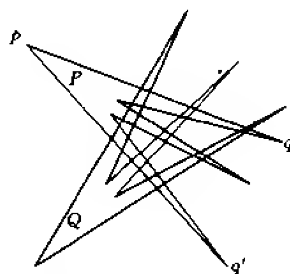


图 5-10　两个星形多边形的交

5.2.2 星形多边形交的算法

图 5-10 中所示的是两个星形多边形 P 与 Q，它们的交不是一个凸多边形，而是由 9 个凸多边形组成的并集。$Z_{5\text{-}2}$ 算法可用来计算星形多边形的交。

对图 5-10 所示的星形多边形应用 $Z_{5\text{-}2}$ 算法时，一般都是插入两条边，要判断新插入的两条边(称为边组)，与数据结构 T 中(未被删去的边和边组与新插入的边组构成相邻关系)的边或边组是否相交，如果相交，则求出交点。待扫描结束时，全部交点已求出并构成交点集。再从任一交点出发，沿多边形边(按逆时针方向)行进，交替找出关联的边与交点，直至回到起始交点，便得到一个交(凸多边形)。继续上述工作，直到交点集为空。

$Z_{5\text{-}3}$ 算法(星形多边形交的算法)

begin

步 1　按 x，y 坐标词典式地分类 $n+m$ 个端点，并将它们放入 E 中。

步 2　**while**　$E\neq\varnothing$　**do**

　　begin

　　　$p\leftarrow$MIN(E)/说明与 $Z_{5\text{-}2}$ 类似/

　　　if　p 是左端点 $\wedge q,q'$ 分别为右端点 $\wedge q.y>q'.y$　**then**

　　　begin

　　　　$s\leftarrow p$ 是左端点且 q 为右端点的线段；

　　　　$s'\leftarrow p$ 是左端点且 q' 为右端点的线段；

　　　　INSERT(s,T)；

　　　　INSERT(s',T)；

　　　　$s_1\leftarrow$ABOVE(s,T)；/ABOVE(s,T)为 s 上方另一多边形正扫描的边或边组/

　　　　$s_2\leftarrow$BELOW(s,T)；/BELOW(s,T)为 s 下方另一多边形正扫描的边或边组/

　　　　if　s_1 与 s 相关　**then**　求交点，并记录交点；

　　　　if　s_2 与 s 相关　**then**　求交点，并记录交点；

```
s'_1←ABOVE(s',T);/ABOVE(s',T)为 s' 上方另一多边形正扫描的边或
                   边组/
s'_2←BELOW(s',T);/BELOW(s',T)为 s' 下方另一多边形正扫描的边或
                   边组/
if  s'_1 与 s' 相交  then  求交点,并记录交点;
if  s'_2 与 s' 相交  then  求交点,并记录交点;
交点按 x,y 坐标词典序插入 E
end
else  if  p 是右端点  then
   begin
     s←p 是右端点的线段;
     DELETE(s,T)
   end
   else  if  p 是交点  then
      begin
        DELETE(位于 p 左侧的两条线段,T);
        将 p 右侧两条线段在 T 中交换位置,寻找新的相邻关系及交点,
        并记录交点。交点按词典序插入 E
      end
end
```

步 3 从任一交点出发,按逆时针方向沿多边形边(该边前进方向上另一端点在另一多边形内部或该边前进方向上有偶数个交点)交替行进,直至回到起始交点,便得到一个交。重复该过程,直至交点集为空。

end

$Z_{5\text{-}3}$算法中"p 左侧的两条线段"是指以 p 为右端点的两条线段,"p 右侧两条线段"意指以 p 为左端点的两条线段。

如果 P、Q 都有 n 个顶点,而且 P 的每条边和 Q 的每一条边相交,那么 $P\cap Q$ 有 n^2 个顶点。假设求一个交点(即交的顶点)需要 1 个单位时间,则求 $P\cap Q$ 耗费 $O(n^2)$时间。由此,求解多边形隐藏线问题,在最坏情况下也一定需要 $O(n^2)$时间。

如果仅需要知道 P 与 Q 是否相交,那么只需要时间 $O(n\log n)$。

5.2.3 任意简单多边形交的算法

两个任意简单多边形交,如图 5-11 所示,比两个凸多边形交或两个星形多边形交都要复杂些,仍设多边形的顶点序列按逆时针方向排列,$P=\{p_1,p_2,\cdots,p_n\}$,$Q=\{q_1,q_2,\cdots,q_m\}$。对 $Z_{5\text{-}3}$算法稍作修改便可用来计算两个任意简单多边形的交。

计算星形多边形交的 $Z_{5\text{-}3}$算法中,当扫描到事件点 p_i 时,如果 p_i 为两条边 s_1 与 s_2 的左端点,则插入该两条边到 T 中;如果 p_i 是两条边 s_1、s_2 的右端点,那么从 T 中删去两条边 s_1 与 s_2。p_i 也可能是一条边的左(右)端点。对于任意简单多边形,每次扫描到新事件点

p_i 时，p_i 也可能是一条边的左端点，或者是两条边的左端点；另外，扫描到事件点 p_j 时，p_j 可能也是一条边或两条边的右端点。因此在 $Z_{5\text{-}3}$ 算法中可以增加"与点 p_i、p_j 关联的是一条边还是两条边(即以 p_i、p_j 为左、右端点的线段是一条还是两条)"的判断，这只要比较 $p_{i-1,x}$、$p_{i,x}$ 与 $p_{i+1,x}$。当 $p_{i-1,x}>p_{i,x}>p_{i+1,x}$ 时，以点 p_i 为左端点的边只有一条边，而当 $p_{i,x}<p_{i-1,x}\wedge p_{i,x}<p_{i+1,x}$ 时，以点 p_i 为左端点的边有两条边。类似判断 p_j 是一条边还是两条边的右端点。算法描述如下。

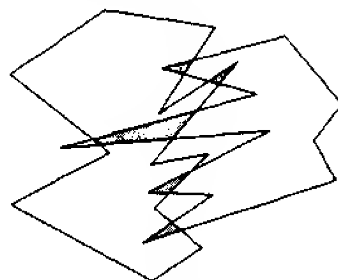


图 5-11　两个任意简单多边形的交

$Z_{5\text{-}4}$ 算法(计算两个任意多边形交的算法)：

begin

步 1　按 x,y 坐标词典式地分类 $n+m$ 个顶点，并把它们放入 E。

步 2　**while**　$E\neq\varnothing$　**do**

begin

$p_i\leftarrow$MIN(E)/说明与 $Z_{5\text{-}2}$ 类似/

if　$p_{i,x}<p_{i-1,x}\wedge p_{i,x}<p_{i+1,x}\wedge p_{i-1,y}>p_{i+1,y}\wedge p_i$ 是左端点　**then**

begin

$s\leftarrow\overline{p_{i-1}p_i}$，$s'\leftarrow\overline{p_ip_{i+1}}$；

INSERT(s,T)；

INSERT(s',T)；

$s_1\leftarrow$ABOVE(s,T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

$s_2\leftarrow$BELOW(s,T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

if　s_1 与 s 相交　**then**　求交点，并记录交点；

if　s_2 与 s 相交　**then**　求交点，并记录交点；

$s_1'\leftarrow$ABOVE(s',T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

$s_2'\leftarrow$BELOW(s',T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

if　s_1' 与 s' 相交　**then**　求交点，并记录交点；

if　s_2' 与 s' 相交　**then**　求交点，并记录交点；

交点按词典序插入 E

end

else　**if**　$p_{i+1,x}<p_{i,x}<p_{i-1,x}\wedge p_i$ 是左端点　**then**

begin

$s\leftarrow\overline{p_{i-1}p_i}$；

INSERT(s,T)；

$s_1\leftarrow$ABOVE(s,T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

$s_2\leftarrow$BELOW(s,T)；/与 $Z_{5\text{-}3}$ 算法中说明相同/

if　s_1 与 s 相交　**then**　求交点并记录交点；

if　s_2 与 s 相交　**then**　求交点并记录交点；

交点按词典序插入 E

end

else if p_i 是右端点 $\wedge p_{i-1,x}<p_{i,x}\wedge p_{i-1,x}<p_{i,x}\wedge p_{i+1,y}>p_{i-1,y}$ **then**

begin

$s\leftarrow\overline{p_ip_{i+1}}, s'\leftarrow\overline{p_{i-1}p_i}$;

DELETE(s,T);

DELETE(s',T)

end

else if p_i 是交点 **then**

begin

DELETE(以 p_i 为右端点的两条线段,T);

将 p_i 为左端点的两条线段在 T 中交换位置,寻找新的相邻关系及交点,并记录交点。交点按词典序插入 E。

end

end

步 3 从任一交点出发,按逆时针方向沿多边形边(该边前进方向上另一端点在另一多边形内部或该边前进方向上有偶数个交点)交替行进,直至回到起始交点,便得到一个交。重复该过程,直至交点集为空。

end

假设两个简单多边形 P、Q 的顶点数均为 n,P 与 Q 边的交点数可以达到 n^2 个,所以 $|E|=n^2+2n$。算法 $Z_{5\text{-}4}$中步 1 耗费 $O(n\log n)$时间,步 2 while 语句最多执行 n^2+2n 次,每次执行 while 语句需要 $O(\log n)$时间,所以步 2 需要 $O(n^2\log n)$时间。步 3 的时间耗费不超过 $O(n^2)$。因此 $Z_{5\text{-}4}$算法的时间复杂性为 $O(n^2\log n)$。

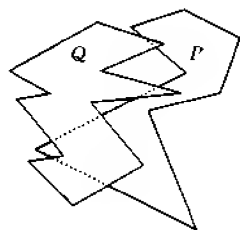


图 5-12　多边形 Q 遮蔽多边形 P

稍微修改 $Z_{5\text{-}4}$算法中的步 3,便可以用于消去隐藏线,具体方法如下:在已求得的交域边界上,从任一交点出发,按逆时针方向沿 P、Q 边交替行进,若沿 P 边行进,则将 P 边改为点线,沿 Q 边行进,Q 边仍为实线,交点是 P 边与 Q 边交替处,直至回到起始交点。重复该过程,直至交点集为空。这样得到的图形是多边形 Q 遮蔽多边形 P 的图形,如图 5-12 所示。上述过程称为算法 $Z'_{5\text{-}4}$。

5.3 半平面的交及其应用

本节介绍求半平面交的算法,两个变量的线性规划问题及算法。

5.3.1 半平面的交

平面上给定 n 个半平面,它们由下述不等式确定

$$a_ix+b_iy+c_i\leqslant 0,\quad i=1,2,\cdots n$$

目的是要求出这n个半平面的交。显然，所要求的交是一个凸多边形区域。上述的n个不等式称为约束条件，满足n个不等式的解（即点）称为可实现解，并且它们的轨迹称为可实现区域（凸多边形区域），图 5-13 中的阴影部分是可实现区域，虚线表示多余的约束。

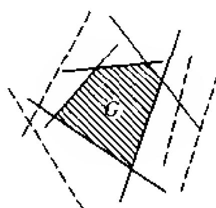


图 5-13 线性约束下的可实现区域

假设已求得前i个半平面的交，它是一个至多i条边的凸多边形区域A_i。A_i与第$i+1$个半平面的交是用线$a_{i+1}x+b_{i+1}y+c_{i+1}=0$切割$A_i$，并且保留右边部分或左边部分得到的，这个工作需要$O(i)$时间。$i$从 2 增至$n$时便可求得可实现域，其时间复杂性为$\sum_{2}^{n} i = O(n^2)$。

利用分治法也可以求解这个问题。设平面上给定n个半平面H_i，$i=\overline{1,n}$，要求构造它们的交

$$H_1 \cap H_2 \cap \cdots \cap H_n$$

借助结合律，可以写成

$$(H_1 \cap \cdots \cap H_{n/2}) \cap (H_{\frac{n}{2}+1} \cap \cdots \cap H_n)$$

两个括号分别表示$\frac{n}{2}$个半平面的交，它们至多是$\frac{n}{2}$条边的凸多边形区域A与A'。两个k条边的凸多边形区域能交$O(k)$次，所以合并A与A'成一个凸多边形区域需要时间$O(n)$。

求n个半平面H_i交的分治算法

输入 由有向线段$\overrightarrow{s_i}$确定的n个半平面H_i，$i=\overline{1,n}$。

输出 凸多边形区域$H=H_1\cap H_2\cap\cdots\cap H_n$。

步 1 将n个半平面H_i分成两个大小近似相等的集合。

步 2 递归地构造凸多边形区域A与A'。

步 3 合并A与A'成H。

令$T(n)$表示用分治算法构造n个半平面的交所需要的时间，则有

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

解得

$$T(n) = O(n\log n)$$

因此，用$O(n\log n)$时间可以构造n个半平面的交，而且这是最优的。

5.3.2 两个变量的线性规划

二维线性规划问题：

目标函数 $f(x,y)=ax+by\rightarrow$最小（或最大） (5-1)

约束条件 $a_ix+b_iy+c_i\geqslant 0$，$i=1,2,\cdots,n$；

$x\geqslant 0, y\geqslant 0$ (5-2)

线性规划问题是在取正值或 0 值的两个变量(x,y)的n个联立线性不等式的约束条

件下，求式(5-1)的最小(或最大)值。线性规划问题的可实现区域是满足式(5-2)中约束的点(x,y)的集合，而且是 n 个半平面的交。可实现区域显然是一个凸多边形区域，该凸多边形的顶点使目标函数减至最小或增至最大。

求解线性规划问题已有许多方法，这里提出的算法是依据分治思想设计的非数值算法。即首先将约束条件根据其斜率及在 x 和 y 轴上的截距进行分类，然后删去每一类中的多余约束条件，并求少量剩余约束条件对应的线链，最后求 3 条线链的交点，便可求得最优值。这种方法的优点在于不需要检验每个约束条件(多余的约束条件较早就被删去)；求直线交点的次数达到最少，一般对各类线簇只需求少数几个交点；一般情况下，不必构造可实现区域多边形的所有边界；函数值的求值次数也达到最少。

$Z_{5\text{-}5}$算法(二维线性规划问题的非数值算法)

输入 目标函数 $f(x,y)=ax+by$

约束条件 $a_ix+b_iy+c_i\geqslant 0,\quad i=1,2,\cdots,n;$

$x\geqslant 0,y\geqslant 0$

输出 使 $f(x,y)$最小(最大)的 x,y 的值及 $f(x,y)$的值

步 1 作线性变换：$Y=ax+by,X=x$(a,b 不同时为 0，设 $b\neq 0$)

经上述变换后，原问题及约束条件为

使 Y 最小；

约束条件：$\alpha_iX+\beta_iY+c_i\geqslant 0\quad i=1,2,\cdots,n;$

$$\alpha_i=a_i-b_i\frac{a}{b};\beta_i=\frac{b_i}{b}$$

步 2 改写约束条件：

$Y\geqslant -\frac{\alpha_i}{\beta_i}X-\frac{c_i}{\beta_i}$(设 $\beta_i\neq 0$)或 $Y\leqslant\frac{\alpha_i}{\beta_i}X+\frac{c_i}{\beta_i}$，$Y>0$，分别令 $X=0,Y=0$，求约束条件在 Y 轴和 X 轴上的截距 $d_i^Y=-c_i/\beta_i,d_i^X=-c_i/\alpha_i$

步 3 **while** $Y\geqslant -\frac{\alpha_i}{\beta_i}X-\frac{c_i}{\beta_i}$ **do** 依据斜率 $e_i=-\frac{\alpha_i}{\beta_i}$的正负号及 Y 轴上截距 $d_i^Y=-\frac{c_i}{\beta_i}$的值把约束条件分类

步 3-1 **if** $e_i>0,(i=\overline{1,m_1})\land$对应的 Y 轴上截距 $d_i^Y>0,(i=\overline{1,m_1})$

then 按递增序排列 $d_i^Y:d_1^Y,d_2^Y,\cdots,d_{m_1}^Y$；

按递增序排列 $e_i:e_1',e_2',\cdots,e_{m_1}'$。

goto 步 3-2

else goto 步 3-6

步 3-2 **if** $e_{m_1}'=e_{m_1}$ **then** $L_1=l_{m_1}$，删去 $l_1,l_2,\cdots,l_{m_1-1}$(见引理 5-1)

goto 步 3-6

else if $e_{m_1}=e_j'\land(e_1',e_j')$与$(d_{e_1'}^Y,d_{m_1}^Y),\cdots,(e_{j-1}',e_j')$与$(d_{e_{j-1}'}^Y,d_{m_1}^Y)$未构成反序

then 删去斜率为 e_1',e_2',e_{j-1}'的直线。

if $d_{e_{m_1}}^Y=d_k^Y$ **then** 删去截距为 $d_1^Y,\cdots,d_{k-1}^Y$的直线。

步 3-3 **while** $e_{m_1}=e_j'$ **do**

步 3-4 **for** $i \leftarrow 1$ **to** $m_1 - j$ **do** 求 l_{e_i} 与(未删去)$l_{e_{j+i}}$ 的交点 p_i。

步 3-5 求 $p_1(X), p_2(X), \cdots, p_{m_1-j}(X)$ 的最小值，设为 $p_{j'}(X)$。$p_{j'}$ 是 l_{e_j} 与 $l_{e_{j+j'}}$ 的交点，删去 e'_j 与 $e'_{j+j'}$ 之间的元素。$j \leftarrow j + j'$，重复步 3-4，步 3-5，直至 $j = m_1$。直线链 $L_1 = l_{d^Y_{m_1}} P_j l_{j+j'} \cdots$。

步 3-6 **if** $e_i < 0 (i = \overline{1, m_2}) \wedge$ 对应的 Y 轴上截距 $d^Y_i > 0, (i = \overline{1, m_2})$

then 按递增序排列 $d^Y_i : d^Y_1, d^Y_2, \cdots, d^Y_{m_2}$；

按递增序排列 $e_i : e'_1, e'_2, \cdots, e'_{m_2}$。

goto 步 3-7

else goto 步 4

步 3-7 **if** $e'_{m_2} = e_{m_2}$ **then** $L_2 = l_{m_2}$，删去 $l_1, l_2, \cdots, l_{m_2-1}$。

goto 步 4

else if $e_{m_2} = e'_j \wedge (e'_1, e'_j)$ 与 $(d^Y_{e_1}, d^Y_{m_2}), \cdots, (e'_{j-1}, e'_j)$ 与 $(d^Y_{e_{j-1}}, d^Y_{m_2})$ 未构成反序

then 删去斜率为 $e'_1, \cdots e'_{j-1}$ 的直线。

if $d^Y_{e_{m_2}} = d^Y_k$ **then** 删去截距为 $d^Y_1, \cdots, d^Y_{k-1}$ 的直线。

步 3-8 用类似于步 3-3 至步 3-5 的方法处理斜率序列中 $e'_{j+1}, e'_{j+2}, \cdots, e'_{m_2}$，得到直线链 L_2。

步 4 **if** $e_i > 0 (i = \overline{1, m_3}) \wedge$ 对应的 X 轴上截距 $d^X_i > 0, (i = \overline{1, m_3})$

then 按递增序排列 $d^X_i : d^X_1, d^X_2, \cdots, d^X_{m_3}$；

按递增序排列 $e_i : e'_1, e'_2, \cdots, e'_{m_3}$。

goto 步 4-1

else goto 步 5

步 4-1 **if** $e'_{m_3} = e_1$ **then** $L_3 = l_1$，删去 $l_2, \cdots, l_{m_3}$。

goto 步 5

else if $e_1 = e'_j \wedge (d^X_1, d^X_{e_1})$ 与 $(e'_1, e'_j), \cdots, (d^X_1, d^X_{e_{j-1}})$ 与 (e'_{j-1}, e'_j) 构成反序

then 删去斜率为 $e'_1, e'_2, \cdots, e'_{j-1}$ 的直线。

if $d^Y_{e_{m_3}} = d^Y_k$ **then** 删去截距为 $d^Y_{k+1}, \cdots, d^Y_{m_3}$ 的直线。

步 4-2 用类似于步 3-3 至步 3-5 的方法处理斜率序列中 $e'_{j+1}, e'_{j+2}, \cdots, e'_{m_3}$，得到直线链 L_3。

步 5 求 L_1, L_2, L_3 的交点，设为 k_1, k_2。从 Y 轴上截距最大的 L_i(比如 L_1)开始，沿 L_1 至与 L_1 相交的交点(L_1 与 L_2 的交点)k_1，在 k_1 处沿 L_2 至与 L_2 相交的交点 k_2，在 k_2 沿 L_3。L_1, k_1, L_2, k_2, L_3 组成直线链 L。

步 6 求 L 中 Y 坐标最小的交点，设为(X_0, Y_0)。

步 7 类似于步 3 至步 6 的方法求出满足约束条件 $Y \leqslant e_i X + d_i$ 的直线链 L'，求 L' 中 Y 坐标最大的交点，设为(X'_0, Y'_0)。

步 8 **if** $Y_0 \leqslant Y'_0$ **then** Y_0, Y'_0 即所求最小、最大值。

else 无解

步 9 用逆线性变换 $x=X, y=(Y-aX)/b$，由 (X_0,Y_0)、(X_0',Y_0') 求出 xy 坐标平面上最小、最大值点。

下面分析 $Z_{5\text{-}5}$ 算法的时间复杂性。

引理 5-1 给定 x,y 平面上 4 个半平面 $y_i \geq a_i x+b_i$，其边界线为 l_i，l_i 的斜率 $a_i>0$，$i=1,2,3,4$。另设 a_1,a_2,a_3 与 a_4 呈递增序，其对应的截距（y 轴上）按递增序排列为 $b_4<b_1<b_3<b_2$。则 l_2 与 l_3，l_3 与 l_4 相交于第 I 象限，且 l_2 与 l_1 在第 I 象限不相交。

证明 观察图 5-14，并由简单几何知识即可证明。

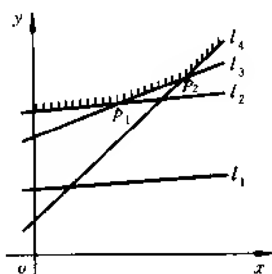


图 5-14 $a_i>0$ 及 y 轴上截距大于 0 时直线间的关系

比较引理 5-1 中的两个序列：

$$a_1,a_2,a_3,a_4 \tag{5-3}$$

$$b_4,b_1,b_3,b_2 \tag{5-4}$$

以截距最大的 b_2 为基准，在式(5-3)中找出相应的 a_2，与 a_2 右邻的元素是 a_3，而 a_3 相应的 b_3 却位于 b_2 的左侧，即 (a_2,a_3) 与 (b_3,b_2) 构成反序，其对应的直线 l_2 与 l_3 必相交，设交点为 p_1。再观察式(5-3)中 a_3，a_3 的右邻是 a_4，而 a_4 相应的 b_4 却位于 b_3 的左侧，(a_3,a_4) 与 (b_4,b_3) 又构成反序，故其对应的直线 l_3 与 l_4 相交，设交点为 p_2。这样，l_2、p_1、l_3、p_2、l_4 组成线链 L_1，L_1 是图 5-14 中阴影区域的边界线。另外，(a_1,a_2) 与 (b_1,b_2) 没有构成反序，故 l_1 与 l_2 不可能在第 I 象限相交，因此 l_1 是多余的约束条件，应删去。

引理 5-2 给定 x,y 平面上 3 个半平面 $y_i \geq a_i x+b_i$，其边界线为 l_i，l_i 的斜率 a_i 均 <0，$i=1,2,3$。另设 a_1,a_2,a_3 呈递增序，其对应的截距（y 轴上）按递增序排列为 b_3,b_1,b_2。则 l_2 与 l_3 相交于第 I 象限，而 l_2 与 l_1 在第 I 象限不相交。

证明 观察图 5-15 所示，并由简单几何知识即可证明。图 5-15 中 l_1 是多余的约束条件，故应删去。

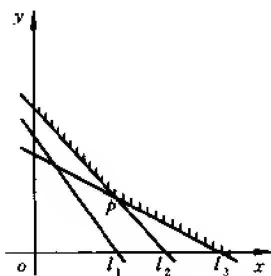


图 5-15 $a_i<0$ 及 y 轴上截距大于 0 时直线间的关系

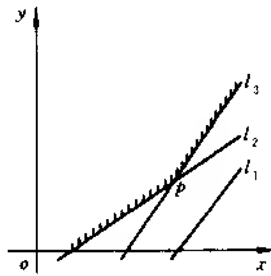


图 5-16 $a_i>0$ 及 x 轴上截距大于 0 时直线间的关系

引理 5-3 给定平面上 3 个半平面 $y_i \geq a_i x+b_i$，其边界线为 l_i，l_i 的斜率 a_i 均 >0，$i=1,2,3$。另设 a_2,a_1,a_3 呈递增序，其对应的截距（x 轴上）按递增序排列为 b_2,b_3,b_1。则 l_2 与 l_3 相交于第 I 象限，而 l_3 与 l_1 在第 I 象限不相交。

证明 观察图 5-16 所示，并由简单几何知识即可证明。(b_3,b_1) 与 (a_1,a_3) 构成反序，

故删去多余的约束条件 l_1。l_2, p, l_3 构成线链 L_3。

推论 5-1 条件与引理 5-1 相同，a_1, a_2, a_3, a_4 呈递增序，其对应的截距（y 轴上）按递增序排列为 b_i, b_j, b_k, b_4；$i, j, k=1,2,3$，并且 $i \neq j \neq k$。则 l_4 与 l_1, l_2, l_3 在第 I 象限均不相交。此时 $L_1=l_4$，删去 l_1, l_2 与 l_3。

推论 5-2 条件与引理 5-2 相同，a_1, a_2, a_3 呈递增序，其对应的截距（y 轴上）按递增序排列为 b_i, b_j, b_3；$i, j=1,2$，并且 $i \neq j$。则 l_3 与 l_2, l_1 在第 I 象限均不相交。此时 $L_2=l_3$，删去 l_2 与 l_1。

推论 5-3 条件与引理 5-3 相同，a_1, a_2, a_3 呈递增序，其对应的截距（x 轴上）按递增序排列为 b_3, b_i, b_j；$i, j=1,2$ 并且 $i \neq j$。则 l_3 与 l_1, l_2 在第 I 象限均不相交。此时 $L_3=l_3$，删去 l_1 与 l_2。

定理 5-1 设二维线性规划问题中 n 个约束条件 $a_i x+b_i y+c_i \geqslant 0$，$(i=\overline{1,n})$（另外，$x \geqslant 0, y \geqslant 0$）所对应的直线在 xy 平面的第 I 象限呈均匀分布，那么 $Z_{5\text{-}5}$ 算法正确地求得了二维线性规划问题的最大、最小值，并且 $Z_{5\text{-}5}$ 算法在最坏情况下的时间复杂性为 $O(n^2)$ 次比较与 $O(n^2)$ 次四则运算，但可以达到 $O(n\log n)$。如果不计分类复杂性，则可以达到 $O(\log n)$。

证明 线性规划问题的可实现区域是满足 n 个约束条件的解（即平面点）的集合，而且它是 n 个半平面的交，可实现区域是一个凸多边形区域，设为 G。G 的边数小于或等于 n。当 G 的边数等于 n 时，n 个约束条件所代表的半平面的边界（直线）都是 G 的边，这是 $Z_{5\text{-}5}$ 算法的最坏情况。最坏情况发生时没有一个约束条件是多余的；否则便有多余的约束条件由 $Z_{5\text{-}5}$ 算法中的步 3-2，步 3-5，步 3-7，步 3-8，步 4-1，步 4-2 等删去，从而节省了大量的计算时间。上述步骤之所以能删去多余的约束条件，主要是由于步 2 与步 3 中 while 语句所做的准备工作，该工作将 n 个约束条件所表示的直线分成 6（或 4）类：$Y \geqslant -(\alpha_i/\beta_i)X - c_i/\beta_i$，$Y>0$ 时，有 3 类，即分别始于步 3-1、步 3-6、步 4 所处理的情况（对应于引理 5-1、引理 5-2、引理 5-3 所讨论的情况）；$Y \leqslant e_i X+d_i$，$Y>0$ 时，也有 3 类，即步 7 所处理的情况。$Z_{5\text{-}5}$ 算法中步 3-1 至步 3-5 处理了第一类约束条件（即 $e_i>0$ 并且 $d_i^Y>0, i=\overline{1,m_1}$）。其方法是，首先将 d_i^Y, e_i 按递增序排列，依据引理 5-1 及推论 5-1，步 3-2 与步 3-5 删去多余的约束条件，并准确判断哪两条直线相交，如果相交，求出交点；然后求出第一类约束条件中所有相交的直线，得到直线链 L_1。$Z_{5\text{-}5}$ 算法中步 3-6 至步 3-8 处理第二类约束条件，得到直线链 L_2；步 4 处理第三类约束条件，得到直线链 L_3。步 5 求出 L_1, L_2, L_3 的交点，得到 G 的下半部边界 L。在构造 L 的过程中，可以增加比较链 L 中直线交点的 Y 坐标值。设这些交点为 $u_1, u_2, \cdots, u_{n/2}$，由于 G 是凸多边形，所以只要发现 $u_i(Y)<u_{i+1}(Y)$，则停止构造 G 的下半部边界 L。$u_i(Y)$ 经步 9 的逆线性变换后便得到最小值点 u_i 的 x, y 坐标，点 u_i 的 Y 坐标值 $u_i(Y)$ 即所求的最小函数值。该算法的步 1 所作的线性变换的作用是转化问题的形式使函数求值的工作变得简单些。总之，$Z_{5\text{-}5}$ 算法采用分治思想，把约束条件分成 6（或 4）个类，既考虑了每个约束条件，又删去了每个类中多余的约束条件，然后逐步构造 G 的下半部边界 L，因此正确地求得问题的解。

$Z_{5\text{-}5}$ 算法的步 1，2 和步 9 需要线性次四则运算。$Z_{5\text{-}5}$ 算法用 $2n$ 次比较可以完成步 3-1，步 3-6，步 4 及步 7 中的分类工作。步 3-1，步 3-6 与步 4 中的排序分别耗费 $O(m_1\log m_1)$，

$O(m_2\log m_2)$与$O(m_3\log m_3)$次比较。步 3-2 用 $4m_1$ 次比较可以删去多余的 $l_1, l_2, \cdots, l_{m_1-1}$，以及斜率为 $e_1', e_2', \cdots, e_{j-1}'$ 的直线和截距为 $d_1^Y, \cdots, d_{k-1}^Y$ 的直线。步 3-4 至多求 $m_1 - j$ 次交点，步 3-5 用 $m_1 - j - 1$ 次比较求出 $p_j(X)$，然后至步 3-4 的循环至多执行 $m_1 - j$ 次。故步 3-2 至步 3-5 需要的比较次数为

$$4m_1 + (m_1 - j - 1)(m_1 - j) = O(m_1^2)$$

算术运算次数

$$C(m_1 - j)(m_1 - j) = O(m_1^2)$$

其中 C 是求一个交点所需要的算术运算次数。

同理，步 3-7 至步 3-8、步 4-1 至步 4-2 分别耗费相同的比较和算术运算次数。由于已给出 n 条直线呈均匀分布，所以 $m_1 = m_2 = m_3 = n/6$，故上述耗费不超过 $O(n^2)$ 次比较，$O(n^2)$ 次算术运算。

链 L_2 的两个端点 L_2^a, L_2^b 分别位于 Y 轴和 X 轴上，L_2 上的顶点的 Y 坐标值呈递减序排列（见图 5-15）。链 L_1 上的顶点的 Y 坐标值呈递增序排列。当 L_2 在 Y 轴上截距小于 L_1 在 Y 轴上截距时，L_1 与 L_2 不相交（删去 L_2，判断 L_3 与 L_1 是否相交）；否则 L_1 与 L_2 相交，只有 1 个交点，设为 k_1。为了求出 k_1，只要判断 L_1 中哪条线段的两个端点分别位于 L_2 与 X、Y 轴组成的多边形内部和外部（假设$\overline{L_1^aL_1^b}$是所求的线段），这个工作在最坏情况下需要 $O(m_1m_2)$ 次比较；然后连接 L_1^a 和 L_2^a，再用同样的方法耗费 $O(m_1m_2)$ 次比较在 L_2 中寻找与$\overline{L_1^aL_1^b}$相交的线段，设为$\overline{L_2^aL_2^b}$。求$\overline{L_1^aL_1^b}$与$\overline{L_2^aL_2^b}$的交点，即 k_1。类似方法求 L_1 与 L_3 的交点 k_2，需要 $O(m_1m_3)$ 次比较。此时得到链 $L = L_2k_1L_1k_2L_3$。如果 L_2 被删去，则只需求 L_1 与 L_3 的交点 k_3，此时得到链 $L = L_1k_3L_3$，其耗费为 $O(m_1m_3)$ 次比较。

如果链 L 由 L_1 开始，则 L_1 的第 1 个顶点即为所求的最小值点，此时耗费 1 次比较。如果链 L 始于 L_2，则 k_1 为所求的最小值点，也只耗费 1 次比较，所以步 6 的耗费为常数次比较。因此步 3 至步 6 所需要的比较次数为

$$O(m_1^2) + O(m_2^2) + O(m_3^2) + O(m_1m_2) + O(m_1m_3) \leqslant O(5n^2/9)$$

由于对称性，步 7 的耗费为 $O(5n^2/9)$。步 8 只需 1 次比较。

总之，$Z_{5\text{-}5}$算法在最坏情况下需要 $O(n^2)$ 次比较和 $O(n^2)$ 次四则运算。如果利用已排序的斜率序列，确定哪些线段相交并求交点，那么耗费线性时间可以确定可实现域的边界。因此 $Z_{5\text{-}5}$算法在最坏情况下的复杂性降低到 $O(n\log n)$。

如果 n 个约束条件所代表的半平面的边界（直线）都是可实现区域 G 的边，那么这是该问题的最坏情况。这时没有一个约束条件是多余的，因而 $Z_{5\text{-}5}$算法中步 3-2，步 3-5，步 3-7，步 3-8，步 4-1，步 4-2 等没有产生实际效益。最坏情况发生时，不必求出 G 的全部边界，只要考虑可实现区域下边界和上边界。对于下边界来说，将斜率排序之后，仅需在序列中找到斜率由负值变为正值的两条直线 l_i 和 l_{i+1}。l_i 与 l_{i+1} 的交点就是要求的最小值点。对于上边界来说，同样处理，只是找斜率由正值变为负值的两条直线的交点，该交点是最大值点。可以用二叉搜索方法寻找这样的两对直线，因此算法的时间复杂性为 $O\left(\log \frac{n}{2}\right)$。如果考虑预处理（分类）时间，则最坏情况时间复杂性为 $O(n\log n)$。 证毕。

另外，利用半平面的交可以设计出求解问题 2-2 的算法，这里就不赘述了。

5.4 多边形的并

本节介绍多边形并的算法。

求两个任意多边形的并是S计算几何中的基本问题之一，该问题的一种特殊情况是求两个矩形的并及多个矩形的并。在超大规模集成电路的辅助设计、数据库中的并发控制等领域中常出现上述问题。因此研究这一问题的快速解法具有实际的应用价值。求两个任意多边形的并的一种方法是逐次判断多边形 P 的每条边是否与多边形 Q 的边相交，这种方法需要 nm 次判断两条线段是否相交。这里提出的算法是基于分治思想设计的，该算法首先求出 P、Q 及 P 与 Q 的凸壳，然后分6种不同情况分别求 $P\cup Q$ 的外围边界及 $P\cup Q$ 内空洞的边界。特别是第6种情况，把 P 与 Q 的凸壳顶点及求出的交点分成两种不同类型来处理。这种方法只需要 $O(n+m)$ 次判断两条线段是否相交，因而减少了所需要的计算时间。另外，该算法比采用求解线性方程组的方法也优越得多。

给定多边形 P 的顶点序列 $p_1,p_2,\cdots,p_n$ 及多边形 Q 的顶点序列 $q_1,q_2,\cdots,q_m$，它们的边界分别为 $\overline{p_1p_2},\cdots,\overline{p_{n-1}p_n},\overline{p_np_1}$ 及 $\overline{q_1q_2},\cdots,\overline{q_{m-1}q_m},\overline{q_mq_1}$。通常，多边形表示边界和内部区域的并。若简单多边形 P 的内部区域是凸集，则此简单多边形是凸的。如果 P 和 Q 是两个任意简单多边形，那么 P 和 Q 的并(表示为 $P\cup Q$)定义为

$$P\cup Q=\{z|z\in P\text{的内域及边界或者}z\in Q\text{的内域及边界}\}$$

下面介绍周培德提出的求 $P\cup Q$ 的一种算法。

$Z_{5\text{-}6}$算法(确定两个任意简单多边形的并的算法)

输入 多边形 P 的顶点序列 $(p_1,p_2,\cdots,p_n)$ 与多边形 Q 的顶点序列 $(q_1,q_2,\cdots,q_m)$，仍设它们按逆时针方向排列。

输出 $(P\cup Q)$ 的顶点序列

步1 求 P 的凸壳 C_1，Q 的凸壳 C_2 及 P 与 Q 的凸壳 C。设 $C_1=\{p_1,p_2,\cdots,p_{i-1},p_i\}$；求 $P'=\{p_i,p_{i+1},\cdots,p_n,p_1\}$ 的凸壳 C_1'，求 P' 与 Q 的凸壳 C'。

步2 **if** $|C_1|=n\wedge q_1,q_2,\cdots,q_m$ 全在 C_1 内

then $P\cup Q=P$，终止。

步3 **if** $C=\{p_i,p_{i+1},\cdots,p_j,q_{i'},q_{i'+1},\cdots,q_{j'}\}\wedge\{p_1,p_2,\cdots,p_{i-1},p_{j+1},\cdots,p_n\}$ 中点不在 C_2 内 $\wedge\{q_1,q_2,\cdots,q_{i'-1},q_{j'+1},\cdots,q_m\}$ 中点不在 C_1 内 **then** $P\cup Q=P$ 与 Q，终止。

else if $p_k(k=\overline{1,i-1};\overline{j+1,n})$ 在 C_2 内 $\vee q_{k'}(k'=\overline{1,i'-1};\overline{j'+1,m})$ 在 C_1 内

then 求 $\overline{p_{k-1}p_k},\overline{p_kp_{k+1}}$ 与 Q 边的交点；$\overline{q_{k'-1}q_{k'}},\overline{q_{k'}q_{k'+1}}$ 与 P 边的交点；$\overline{p_{k-1}p_k},\overline{p_kp_{k+1}}$ 与 $\overline{q_{k'-1}q_{k'}},\overline{q_{k'}p_{k'-1}}$ 的交点。从 p_i 出发，沿 P 边行进，碰到交点，改沿 Q 边行进，再碰到交点，改沿 P 边行进，直至回到 p_i(称巡回方法)。此点列即 $P\cup Q$ 的边界点序列。从 Q 外剩余的 P 点出发，用巡回方法找空洞边界点列。终止。如无交点，则 $P\cup Q=P$ 与 Q，终止。

步 4　**if**　$q_1,q_2,\cdots,q_m$ 在 C_1 内 $\wedge|C_1|<n\wedge$(C'代替 C,C_1' 代替 C_1 之后,步 3 的前条件成立)

then　$P\supset Q$,即 $P\cup Q=P$,终止。

else　**if**　C'中轮流出现 P'与 Q 的顶点

then　用步 7 中方法求 P'边与 Q 边的交点。从 C_1 中的 P 边出发,用巡回方法找点序列;再由剩余 p 点出发,用巡回方法找点列(后者为空洞部分)。上述点序列围成的域即所求的并,终止。

如无交点,则 $P\cup Q=P$,终止。

步 5　**if**　$q_1,q_2,\cdots,q_m$ 不在 P 内 $\wedge\, p_1,p_2,\cdots,p_n$ 不在 Q 内 $\wedge|C_1|=n\wedge|C_2|=m\wedge|C|=n+m$

then　在 C 中求相邻的 P、Q 顶点关联的边的交点,用巡回方法找并,终止。

步 6　**if**　$|C_1|=n\wedge|C_2|=m\wedge|C|<n+m$　**then**　沿 C 查顶点

步 6-1　**while**　p_i 与 q_{j+h}(p_{i+l}与 q_j)是 C 中相邻的顶点　**do**

if　与 p_i(p_{i+l})关联的 P 边和与 q_{j+h}(q_j)关联的 Q 边相交

then　求出交点 r_k;$p_ir_kq_{j+h}$($p_{i+l}r_kq_j$)组成子点列　**goto**　步 6-2

else　**if**　与 p_i 关联的点 $p_{i'}$($\in C_1$)首次出现在 Q 内 $\wedge$ 与 q_{j+h}关联的点 $q_{j'+h}$($\in C_2$)首次出现在 P 内

then　与 $p_{i'},q_{j'+h}$关联的边相交,求出交点 r_k;r_k 与 p_i,q_{j+h}等组成子点列:$p_i,\cdots,r_k,\cdots,q_{j+h}$,　**goto**　步 6-2

else　**if**　$p_1,p_2,\cdots,p_{i-1},p_{i+l+1},\cdots,p_n$ 不在 C_2 内 $\wedge\, q_1,q_2,\cdots,q_{j-1},q_{j+h+1},\cdots,q_m$ 不在 C_1 内

then　$P\cup Q=P$ 与 Q,终止。

步 6-2　**if**　$p_i,\cdots,p_{i+l}$(或 $q_j,\cdots,q_{j+h}$)是 C 中连续点列

then　$p_i,\cdots,p_{i+l}$(或 $q_j,\cdots,q_{j+h}$)是 $P\cup Q$ 的部分边界点列

步 6-3　从 C 中 p_i 出发,所有相互连接的子点列组成 $P\cup Q$ 的顶点序列,终止。

步 7　**if**　$|C_1|<n\wedge|C_2|<m\wedge|C|<n+m$　**then**　沿 C 查顶点

步 7-1　$i\leftarrow1$,$C=(v_1,v_2,\cdots,v_h)$,$C'\leftarrow C$,$h<n+m$

步 7-2　**while**　$p_i=v_i\wedge q_j=v_{i+1}$　**do**

if　与 p_i 关联的 P 边和与 q_j 关联的 Q 边相交

then　求出交点 r_k;$p_ir_kq_j$ 组成子点列,**goto**　步 7-3

else　**if**　$p_{i'},q_{j'}$分别是 C_1 与 C_2 中相交边的端点 $\wedge\, p_{i'}$在 C_2 内 $\wedge\, q_{j'}$在 C_1 内

then　判断与 $p_{i'},q_{j'}$关联的 P、Q 边是否相交

if　相交　**then**　求出交点 $r_{k'}$;$p_i,\cdots,r_{k'},\cdots,q_j$ 组成子点列。

else　判断 $p_{i'},q_{j'}$相邻的 P,Q 顶点所关联的边是否相交,如果相交,则求出交点,产生子点列;否则,$P\cup Q=P$ 与 Q,终止。

步 7-3　**if**　r_k 关联的边与 p_i(或 p_{i-1},q_j,q_{j-1})关联的边相交

then　求出交点 $r_{k'}$

if　$r_{k'}$关联的边与 v_{i-1}(或 v_{i+2})关联的边相交　**then**　求出交点 $r_{k''}$

if 与 $r_k(r_{k'}, r_{k''}, r_{k'''})$ 相邻点关联的边相交

then 求出交点，并删去 $r_k(r_{k'}, r_{k''}, r_{k'''})$ 及相邻点

步 7-4 重复步 7-3，直至 r_k 关联的边与 p_i（或 q_j）关联的边不相交 ∨ $r_{k'}$ 关联的边与 v_{i-1}（或 v_{i+2}）关联的边不相交 ∨ 与 $r_k(r_{k'}, r_{k''}, r_{k'''})$ 相邻点关联的边不相交。

步 7-5 **if** 与 p_i（或 q_j）关联的两条边都已处理（已求过交点）

then 删去 p_i（或 q_j）

步 7-6 **if** C' 中有连续的 P 点（或 Q 点）∨ $P(Q)$ 的凹点形成的三角形△内无 $Q(P)$ 的顶点 ∨ △的顶点都在另一多边形的同一侧

then 删去连续点列或△的顶点

步 7-7 $i \leftarrow i+1$

步 7-8 求 P、Q、C 中剩余点集与剩余交点组成的点集的凸壳 C'。

步 7-9 重复步 7-2（找出 C' 中新的相邻的顶点，处理新的相邻顶点）至步 7-8，直至剩余点集为空。

步 8 从 C 中任一点 p_i（或 q_j）出发，所有相互连接的子点列组成 $P \cup Q$ 的外围边界；从剩余的在 Q 外的某 p 点出发，沿与该 p 点关联的 P 边行进，用巡回方法寻找空洞的边界；从剩余的交点 r_k 出发，沿与 r_k 关联的 P 边行进，用巡回方法寻找空洞的边界。如沿 P 边行进一周，没有遇到交点，则 $P \cup Q = P$ 与 Q，终止。

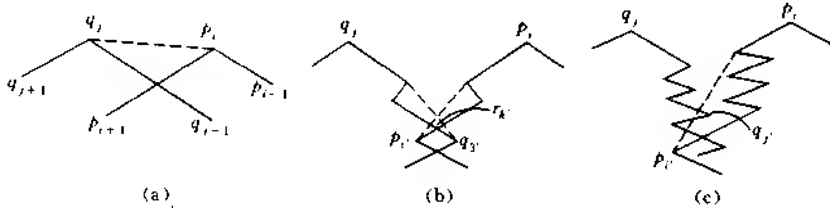
下面分析 $Z_{5\text{-}6}$ 算法的时间复杂性。

定理 5-2 给定平面上两个任意多边形 $P(p_1, p_2, \cdots, p_n)$ 和 $Q(q_1, q_2, \cdots, q_m)$，$Z_{5\text{-}6}$ 算法正确地求出 $P \cup Q$ 的边界顶点序列，而且该算法的时间复杂性为 $O(nm)$ 次比较和 $O(n+m)$ 次判断两条线段是否相交。

证明 平面上两个任意多边形 P 与 Q 的位置关系只有三 种不同情况：P 与 Q 分离（即 P 与 Q）；P 与 Q 相交；$P \supset Q$ 或 $P \subset Q$。这三种不同情况的出现依赖于 P 与 Q 的顶点分布，这是 $Z_{5\text{-}6}$ 算法的依据。

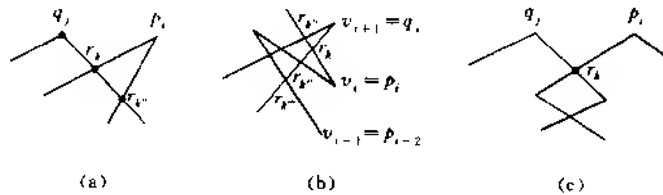
$Z_{5\text{-}6}$ 算法首先求出 P、Q 及 P 与 Q 的凸壳 C_1、C_2 及 C。如果 $C_1 = \{p_1, p_2, \cdots, p_{i-1}, p_i\}$，则还要求出 $P' = \{p_i, p_{i+1}, \cdots, p_n, p_1\}$ 的凸壳 C_1'，P' 与 Q 的凸壳 C'。其作用是在某些情况下将考查 P、Q 的全部顶点转化为考查 C_1、C_2、C、C_1' 及 C' 的顶点，以减少判断线段相交的次数。当 $Z_{5\text{-}6}$ 算法中步 2 的条件成立时，显然有 $P \supset Q$，即 $P \cup Q = P$。如果 P 与 Q 的位置关系属于分离关系，则 P 的顶点不可能位于 C_2 内，Q 的顶点也不可能位于 C_1 内，或者 P、Q 的某些顶点 p_k 和 $q_{k'}$ 分别位于 C_2、C_1 内，但与 p_k、$q_{k'}$ 关联的边不相交。这是步 3 完成的一部分工作。当与 p_k 和 $q_{k'}$ 关联的边相交时，则求出交点，利用巡回方法找出并的边界点列。这是步 3 完成的另一部分工作。如果 Q 的顶点都在 P 内，并且 C' 代替 C、C_1' 代替 C_1 之后，C_1' 中的 P 点不在 C_2 内，C_2 中的 Q 点不在 C_1' 内，则 C_1' 与 C_2 分离，即 $P \supset Q$。否则，如果 C' 中 P 点与 Q 点轮流出现，则用步 7 方法求交点，把问题转化为求 P' 与 Q 的交点。然后用巡回方法找并的边界。这是步 4 的工作。算法中的步 5 处理 P 与 Q 都是凸多边形并且 $|C| = n+m$ 的情况，这时只要求出 C 中相邻的 P、Q 顶点关联的边的交点，然后用巡回方法找出并的边界点列即可。两个凸多边形相交时，至多有 $n+m$ 个交点（步 5 已处理有 $n+$

m 个交点的情况），一般情况有 2 个交点（退化情况时只有 1 个交点），步 6-1 至多循环执行两次便可求出 2 个交点，再按步 6-3 的方法就可以找出 $P\cup Q$ 的边界顶点序列。如果没有交点，则 P 与 Q 分离。难以处理的是相交的 P、Q 为任意简单多边形，这时 P、Q 边界的相交情况可以归纳为步 7-2 与步 7-3 的条件部分所描述的，如图 5-17 与图 5-18 所示。



(a)　　(b)　　(c)

图 5-17　步 7-2 的示意图



(a)　　(b)　　(c)

图 5-18　步 7-3 的示意图

算法的步 7-2 与步 7-3 循环若干次之后便执行步 7-5 与步 7-6，删去已处理过的点。然后求剩余点集的凸壳，再重复步 7-2 至步 7-8 的工作，求出所有相交边界的交点。由于 $Z_{5\text{-}6}$ 算法是逐点考查逐点删去，因此不可能遗漏相交的边。算法中的步 8 求出 $P\cup Q$ 的外围边界的顶点序列及内部空洞的边界顶点序列。总之，$Z_{5\text{-}6}$ 算法正确地求出了 $P\cup Q$ 的边界顶点序列。

$Z_{5\text{-}6}$ 算法中的步 1 耗费不超过 $O((n+m)\log(n+m))$ 次比较。步 2 需要 $O(nm)$ 次比较。步 3 的前半部分可以利用步 2 的结果，不需要新的开销。步 3 的后半部分需要 $2(a+b)<n+m$ 次判断两线段是否相交，其中 $a(b)$ 为 $P(Q)$ 的顶点位于 $Q(P)$ 内的顶点数。步 4 的前半部分耗费不超过 $O(nm)$ 次比较，后半部分不会超过 $O(nm)$ 次比较和 $O(n+m)$ 次判断两条线段是否相交。步 5 需要 $O(nm)$ 次比较和求 $(n+m)$ 次交点。步 6-1 至多需要 $n+m$ 次比较与 $n+m$ 次判断两条线段是否相交，步 6-2 的耗费已包含于步 6-1 之中。步 7-2 至多耗费 $n+m$ 次比较与 $n+m$ 次判断两条线段是否相交。步 7-3 与步 7-4 需要 $C(n+m)$ 次判断两条线段是否相交，其中 C 是常数。步 7-5 与步 7-7 所需时间可以忽略不计。步 7-6 耗费 $O(n)$ 或 $O(m)$ 次比较。执行步 7-8 不会超过 $O((n+m)\log(n+m))$ 次比较。由于每执行步 7-3 至步 7-6 的一次循环可以删去大约比一半还多的顶点与交点，所以步 7-9 的循环执行常数次便可使剩余点集为空。步 8 的时间可以忽略不计。$Z_{5\text{-}6}$ 算法所需要的比较次数为：

$$\max[O((n+m)\log(n+m))+O(nm),O((n+m)\log(n+m))+(n+m),$$
$$O((n+m)\log(n+m))+(n+m)+O(n)+O(m)+O((n+m)\log(n+m))]$$

$=O((n+m)\log(n+m)+O(nm))=O(nm)$

判断两条线段是否相交的次数为：

$$\max[n+m, O(n+m), n+m, (n+m)+c(n+m)]=O(n+m)$$ 证毕。

另外，利用 $Z_{5\text{-}4}$ 算法中步 1、步 2 求出 P 与 Q 的交点，然后修改步 3，可以求出 $P\cup Q$。算法描述如下。

$Z_{5\text{-}7}$算法（计算 $P\cup Q$ 的算法）

步 1 求 P 与 Q 的凸壳，设为 C。

步 2 利用 $Z_{5\text{-}4}$ 算法中步 1、步 2 计算 P 与 Q 的交点，并记录交点于 A 中。

步 3 从凸壳 C 中点 p_i（或 q_j）出发，沿 P 边（或 Q 边）按逆时针方向行进，碰到交点改沿另一多边形边按逆时针方向行进，直至回到出发点，便得到 $P\cup Q$ 的外围边界。

if $A=\varnothing$ **then** 输出结果，终止。

else 从 A 中任一剩余点出发，按顺时针方向沿多边形边（该边前进方向上另一端点不在另一多边形内而且该边前进方向上无交点或奇数个交点，或者该边前进方向上另一端点在另一多边形内并且该边前进方向上有偶数个交点）行进，碰到交点改沿另一多边形边行进，直至回到出发点，便得到 $P\cup Q$ 的空洞边界。该过程继续下去，直至 $A=\varnothing$。输出结果。终止。

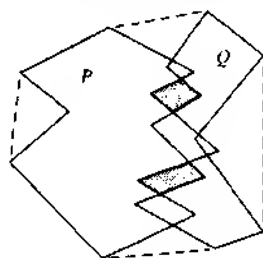


图 5-19 $Z_{5\text{-}7}$算法示意图

图 5-19 是算法 $Z_{5\text{-}7}$的示意图，图中虚线为凸壳 C 的一部分，该图有两个空洞，用阴影表示。

由于 $Z_{5\text{-}4}$算法中步 1、步 2 的时间复杂性为 $O(n^2\log n)$，而 $Z_{5\text{-}7}$算法中步 1、步 3 的时间复杂性不超过此限界，所以 $Z_{5\text{-}7}$算法的时间复杂性为 $O(n^2\log n)$。另外，$Z_{5\text{-}7}$算法中，步 3 else语句之后括号内的条件可以取消。

5.5 凸多面体的交

给定两个凸多面体 P 和 Q，它们分别有 n，m 个顶点，构造 P 和 Q 的交，记为 $P\cap Q$。

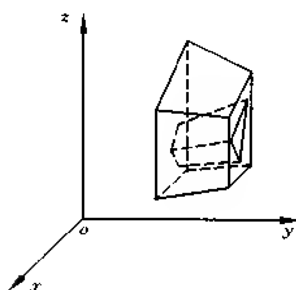


图 5-20 两个切片之间的台状多面体

显然，P 和 Q 的交是凸多面体。计算 $P\cap Q$ 的简单方法是：检查 P 的每个小侧面与 Q 的每个小侧面是否相交，若相交，则记录交，该交为线段；然后由此确定 $P\cap Q$。如果 P 和 Q 的所有小侧面都不相交，那么 $P\cap Q=\varnothing$，即 P 与 Q 是分离的，或者一个多面体在另一个的内部。这种方法的时间复杂性为 $O((n+m)^2)$。

5.2 节中介绍的"梯形交组成 $P\cap Q$ 的算法"可以推广到三维：首先用通过 P、Q 的一个顶点并垂直于 y 轴的平面切割空间，得到 P 或 Q 的一个切片，用同样方法可以得到 $n+m$ 个切片。相邻的两个切片之间形成至多两个台状多面体，共计不超过 $2(n+m-1)$个台状多面体。图 5-20

所示的两个台状多面体，处在内部的台状多面体就是 $P\cap Q$ 的一部分。显然至多有 $n+m-1$ 个切片间的嵌套结构，对每个结构计算 $P\cap Q$ 的一部分。然后计算 $P\cap Q$。这种方法的时间复杂性也为 $O((n+m)^2)$。

5.2 节中的"沿 P 边和 Q 边行进寻找 $P\cap Q$ 的算法"难以推广到三维。

定义 2-6 的思想可以推广到三维：如果(1)P_0 是至多有 4 个顶点的多面体；(2)$P_k=P$；(3)删去 P_i 的某些顶点可以得到 P_{i-1}，则多面体系列 $P_0,P_1,\cdots,P_k$ 是凸多面体 P 的均衡分层表示。

耗费 $O(n)$时间可以构造凸多面体 P 的均衡分层表示。如果 $P_0,P_1,\cdots,P_k$ 是 P 的均衡分层表示，那么 $k=O(\log n)$。

为了构造凸多面体 P 的均衡分层表示，先引入独立集的概念及构造独立集的算法。

多面体的边和顶点投影到 xy 平面上将形成一个平面图。给定平面图 $G=(V,E)$，$I\subset V$，如果 I 中没有两个结点有边连接，则称 I 为 G 的独立集。若$|I|=k$，并且不存在 G 的独立集 I'，$|I'|>k$，则称 I 为 G 的最大独立集。图 5-21(a)是 20 面体投影到 xy 平面上形成的平面图，它的 3 个结点 q_1,q_2,q_3 是平面图的独立集，而且是最大独立集。

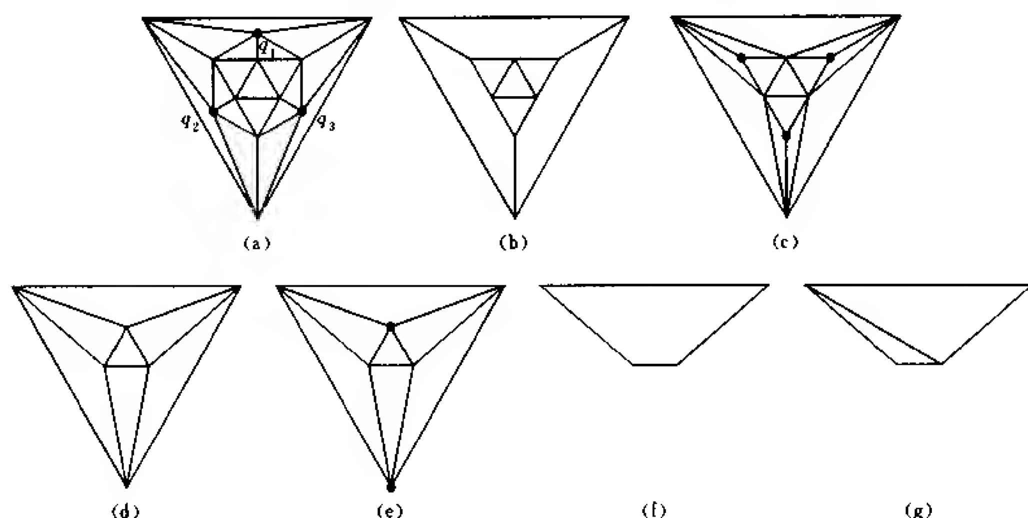


图 5-21　二十面体的 xy 平面图及独立集

构造 P 的均衡分层表示的过程如下：首先构造 P_1。(1)令 $P_0=P$，寻找 P_0 的顶点独立集，如图 5-21(a)所示，$I=\{q_1,q_2,q_3\}$，删去 q_1,q_2,q_3 及与该 3 个点关联的边，如图 5-21(b)所示。每删去一个 $q_i(i=1,2,3)$，便产生一个新的小侧面，该小侧面是五边形，它投影到 xy 平面上是一个四边形(有两条边共线)。(2)对三个四边形分别进行三角剖分，如图 5-21(c)所示，该图对应的三维图就是多面体 P_1，它有 9 个顶点，14 个小侧面，显然 P_1 嵌套在 P_0 的内部。然后构造 P_2，方法与构造 P_1 相同，如图 5-21(d)所示，这次不必三角剖分，该图对应的三维图是多面体 P_2，P_2 有 6 个顶点，8 个小侧面。重复该过程，图 5-21(e)中$|I|=2$，删去 I 后产生图 5-21(f)中的图形，再进行三角剖分，得到图 5-21(g)中的图形，它对应的三维图是四面体，即 P_3。这样便形成了 P 的均衡分层表示：P_0,P_1,P_2,P_3。

注意，选择独立集中的点时，应从度数最低的结点中挑选，其方法如下。

选择独立集的算法

输入 图 $G=(V,E)$

输出 独立集 I

$I\leftarrow\varnothing$

对 G 中度数≥9 的结点进行标记

while 某些结点仍未标记 **do**

选一个未标记结点 v，标记 v 及 v 的所有邻接点

$I\leftarrow I\cup\{v\}$

如果图 G 有 n 个结点，则选择独立集的算法的时间复杂性为 $O(n)$。

构造凸多面体 P 的均衡分层表示的算法

输入 凸多面体 P

输出 $P=P_0,P_1,\cdots,P_k,k=\log n$

$i\leftarrow 0,P_0\leftarrow P$

while $|P_i|>4$（即 P_i 顶点数大于 4） **do**

用"选择独立集的算法"寻找 P_i 的独立集 I。

初始化 $P_{i+1}\leftarrow P_i$

for 每个顶点 $v\in I$ **do**

从 P_{i+1} 中删去 v 及其关联的边，形成新的小侧面 f，f 在 xy 平面上的投影是多边形，如果该多边形的边数大于 3，则进行三角剖分。它对应的三维图即 P_{i+1}。

$i\leftarrow i+1$

可以用图 2-2 所示的树结构表示凸多面体 P 的均衡分层表示，该树深度为 $O(\log n)$。

给定凸多面体 P 与 Q，它们分别有 n、m 个顶点，构造 $P\cap Q$ 的算法如下：

$Z_{5.8}$算法（构造 $P\cap Q$ 的算法）

步 1 构造 P、Q 的均衡分层表示。

步 2 检查 P 的哪些顶点在 Q 内，哪些在 Q 外。如果凸多面体 P 的棱$\overline{p_ip_{i+1}}$的端点 p_i 在 Q 内，而 p_{i+1} 在 Q 外，则记录$\overline{p_ip_{i+1}}$。对 Q 同样处理。

步 3 **if** 小侧面 P_a 的顶点 p_{i-1}、p_i、p_{i+1} 位于 Q 的小侧面 Q_b 的两侧 ∧ 小侧面 Q_b 的顶点 q_{j-2}、q_{j-1}、q_j、q_{j+1} 位于 P 的小侧面 P_a 两侧

then 计算 P_a 与 Q_b 的交线（如图 5-22 中点线所示）

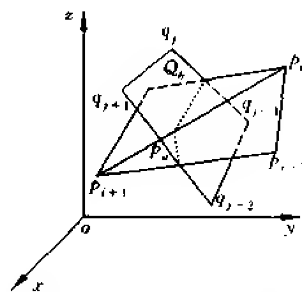


图 5-22 计算小侧面之间的交线（点线）

步 4 利用步 3 找出 P 与 Q 的所有交线，设为 C。

步 5 位于 P 内的 Q 顶点、位于 Q 内的 P 顶点与折线 C 的顶点共同组成 $P\cap Q$ 的顶点。

$Z_{5.8}$算法的步 1 耗费 $O(n+m)$时间，步 2 需要 $O((n+m)\log(n+m))$时间，步 3 只需要常数时间。步 4 的开销依

赖于相交的小侧面数目，而小侧面数目为$O(n)$、$O(m)$，故步4耗费$O(n+m)$时间。步5也不会超过$O(n+m)$时间，所以$Z_{5\text{-}8}$算法的时间复杂性为$O((n+m)\log(n+m))$。

为了实现$Z_{5\text{-}8}$算法，还要设计判断点是否在凸多面体内部的算法等，这些算法都可以由相应的二维算法得到，这里就不赘述了。

$Z_{5\text{-}2}$算法可以推广到三维，用于计算$P\cap Q$。$Z_{5\text{-}2}$算法中的事件点序列仍为凸多面体的顶点。规定凸多面体的每个小侧面的正向法线方向为垂直于小侧面并指向凸多面体外侧方向。过事件点作垂直于y轴的平面，该平面切割P和Q成两个凸多边形，而且一个多边形包含于另一个内部，如图5-20所示。把此嵌套的凸多边形作为与该事件点相关联的信息存储起来，对每个事件点都如此处理，这相当于$Z_{5\text{-}2}$算法中"P是左端点的线段"中的线段，当相邻两切片中P、Q切口(凸多边形)互为相反的包含关系或者包含关系有所改变时，计算P、Q相应小侧面的交线，然后这些交线与位于P内部的Q凸多面体及位于Q内部的P凸多面体构成$P\cap Q$。这一算法称为**$Z_{5\text{-}9}$算法**。

$Z_{5\text{-}9}$算法可以用于一般简单凸多面体的交。凸多面体的小侧面的正向规定为垂直于该小侧面并指向凸多面体外侧的方向。小侧面是凸多边形，其顶点顺序与小侧面正向服从右手法则。小侧面所在的平面与过事件点并垂直于y轴的平面的交是一条线段，由类似的这些线段组成切口凸多边形。

计算$P\cap Q$的另一种方法是由Muller-Preparata(1978)提出的，其思想是：若已知交中的一点p，则通过映射技术可以得到交。设P和Q的内部包含原点，我们把P或Q的每个小侧面f与它所在的平面π_f的一侧相关联，即

$$a(f)x + b(f)y + c(f)z \leqslant 1$$

解释三元组$(a(f),b(f),c(f))$为一个点，则已建立π_f与点之间的一个变换。该变换把离原点距离为l的点映射到离原点距离为$\frac{1}{l}$的平面，反之亦成立。用$\delta(\ \ )$表示该变换，$\delta(p)$是与点p成对应关系的平面，而$\delta(\pi)$是与平面π成对应关系的点。

可以证明，如果P是包含原点的凸多面体，S是P的所有π_f所对应的点的集合，那么P内部的每个点p对应于一个平面$\delta(p)$。

凸多面体P的顶点在xy平面上的投影所形成的点集的凸壳记为P^*。

M-P算法(概要)

步1 构造P^*和Q^*。

步2 计算$P^*\cap Q^*$。

if $P^*\cap Q^*=\varnothing$ **then** $P\cap Q=\varnothing$

else 设$P^*\cap Q^*=D$ 若点$(x,y)\in D$，过点(x,y)作垂直于xy平面的直线l，l位于P内的部分(线段)记为l_P。同样计算l_Q。l_P与l_Q的公共部分(线段)即$P\cap Q$的组成部分。

M-P算法的时间复杂性为$O((n+m)\log(n+m))$。

第6章 矩形几何

由于矩形可以看成是特殊的凸多边形或特殊的多边形，所以第5章关于凸多边形或多边形的交、并算法也可以用于矩形的交、并。但是矩形具有其特殊的性质(它的边是相互垂直或平行的线段)，因此应该设计出满足这种特殊结构的更有效的算法，而且在研究这类问题的过程中可以更进一步了解矩形或正交线段的几何性质，同时还可以描述这些算法能应用的一类问题的特征。

正交线段和矩形是构成许多应用的基础。比如，集成电路制造中所用的掩模常表示为边平行于坐标轴的一组矩形，任务是证实设计规则和要求均能被满足。我们对此任务稍加变换就可以成为一个矩形几何问题(例如矩形交问题)。数据库中由并发控制产生的死锁问题的解决也导致出一个矩形并问题。

本章将着重介绍解决几个矩形几何问题的算法，包括一组垂直、水平线段交的判定算法，计算矩形并的面积的算法，矩形并的周长的算法，计算矩形并的轮廓、闭包、非平凡轮廓和外轮廓以及矩形交的算法等。

6.1 判定垂直、水平线段是否相交的算法

平面上给定 n 条线段 $s_1,s_2,\cdots,s_n$，它们的左端点为 a_i，右端点为 b_i，$s_i(i=\overline{1,n})$ 不是垂直的就是水平的。我们采用5.1节中平面扫描技术判定哪些线段相交，具体做法如下：线段组中的线段之间的关系表示成二叉树，例如图6-1所示的线段组，经过对端点的 x 坐标的分类，次序是

$$<s_1\ s_1><s_4<s_2\ s_2><s_5<s_3\ s_3><s_6>s_4$$

$$<s_7\ s_7>>s_5<s_8\ s_8>>s_6$$

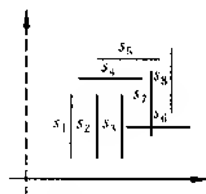


图6-1 线段组

其中“<”表示线段的始点，“>”表示线段的终点。对该线段组的扫描(插入、删去和搜索)过程表示为图6-2所示的二叉树序列。

判定垂直、水平线段是否相交的算法

输入 线段组 $s_1,s_2,\cdots,s_n$ 及端点 a_i,b_i 的 x,y 坐标，$i=\overline{1,n}$

输出 相交的线段对

步1 对线段组按端点的 x 坐标分类，并存入 E。

步2 按 E 中次序逐个将各线段之端点，根据其 x 坐标(水平线)或 y 坐标(垂直线)的大小分成始点和终点。

步3 每当遇到始点时，在二叉树中加入该点，遇到终点时，删去该点。

步4 二叉树中左右子结点的决定是根据这些点的 y 坐标大小来确定的，y 坐标小的为左子结点，大的为右子结点。

步5 遇到垂直线段 s_i 时，始点和终点同时出现，投影是一区间，这时作区间搜索。如

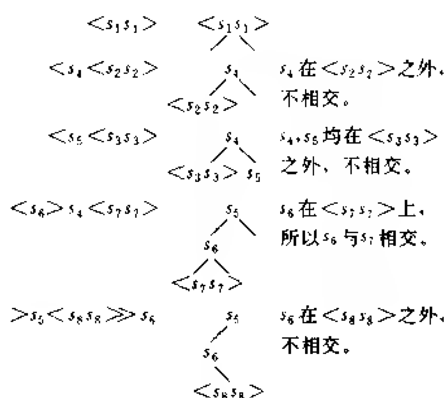


图 6-2　对图 6-1 所示线段组的扫描过程

有一水平线段 s_j 位于该区间内，则 s_i 与 s_j 相交。

该算法的时间复杂性为 $O(n\log n)$。

6.2　矩形几何问题的特征及解决问题的途径

假设平面上给定 n 个矩形 $R_i(i=\overline{1,n})$，R_i 的边分别平行于 x 轴和 y 轴，这样的 n 个矩形称为同等安置的矩形，它们的边可以看成是正交的平行线段的集合。同等安置矩形的特征是把平面划分成条状域，在每个条中只有一个变量，即是一维的。比如考虑垂直条中，所有垂直截面是相同的，它们由和条交叉的水平边的纵坐标组成。如图 6-3 所示的垂直条 $[X[i-1],X[i]]$ 中，垂直截面是由 6 条水平边的纵坐标 y_1,y_2,y_3,y_4,y_5,y_6 组成。与垂直条 $[X[i-1],X[i]]$ 相邻的两个条是 $[X[i-2],X[i-1]]$ 与 $[X[i],X[i+1]]$，它们的截面都可以由修改相邻条的截面而得到。

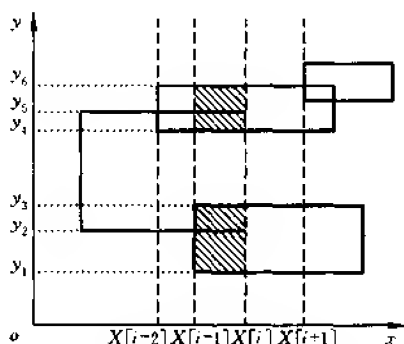


图 6-3　垂直条 $[X[i-1],X[i]]$

矩形并的面积、并的周长、并的轮廓，矩形交等问题具有一个共同的性质：给定问题的解可以和两个半平面（一条垂直线划分平面为两个半平面）的每个半平面中相似子问题的解联系起来。比如，矩形交问题的解是子问题解的并，矩形并的面积、周长等问题的解是子

问题解的算术和，而矩形并的轮廓问题的解是子问题解相应部分的连接等。在这些情况中，平分线(扫描线)和矩形集合的交(扫描线上的一线段)包含了部分解的信息，并且扫描线左半平面得到的解不可修改，它是最后解的一部分，随着扫描线右移，扫描线左侧解得到推广，直至获得整体解。这既是解决矩形几何问题的一条途径，也是平面扫描方法的特性。

由图 6-3 可以看出，垂直截面$[X[i-1],X[i]]$是纵坐标 $y_i(i=\overline{1,6})$的一个序列，而 $y_i(i=\overline{1,6})$又是矩形集合中水平线段的纵坐标分类序列中的一个子序列。因此可以不用优先队列而用线段树就能适用于这里的平面扫描。

用区间$[B(v),E(v)]$(B 为左端点，E 为右端点)和其他参数来描述线段树的每个结点 v 的特征，其中结点计数 $c(v)$表示当前分配给 v 的线段的数目。参数 $c(v)$对所有矩形问题都是需要的，但针对不同矩形问题要增加不同的参数。线段树上的操作包括插入和删去。设$[b,e]$是一个区间，作用于结点 v 的 INSERT$(b,e;v)$如下：

Procedure INSERT$(b,e;v)$

1. **if** $b\leqslant B[v]\wedge E(v)\leqslant e$ **then** $c(v)\leftarrow c(v)+1$
2. **else** **if** $b<\lfloor(B(v)+E(v))/2\rfloor$
   then INSERT$(b,e;\text{LSON}(v))$
3. **if** $\lfloor(B(v)+E(v))/2\rfloor<e$
   then INSERT$(b,e;\text{RSON}(v))$
4. update(v)/修改 v 的指定参数/

对不同具体问题，行 4 将有不同内容。

6.3 矩形并的面积与周长

给定平面上 n 个同等安置的矩形 $R_1,R_2,\cdots,R_n$，定义 n 个矩形的并 $F=R_1\cup R_2\cup\cdots\cup R_n$ 为

$$F=\{p\mid p\in R_1 \text{ 或 } p\in R_2 \text{ 或 } \cdots \text{ 或 } p\in R_n\}$$

定义 F 的周长为边界长，F 的面积是边界所围域的面积。下面介绍利用平面扫描方法计算 F 的面积和周长。

先考虑一维情况。给定数轴上 n 条线段 $L=\{l_1,l_2,\cdots,l_n\}$，其中 $l_1=[a_1,b_1],l_2=[a_2,b_2],\cdots,l_n=[a_n,b_n]$，计算它们并的长度。

计算 $l_1\cup l_2\cup\cdots\cup l_n$ 的长度的算法

步 1 按 l_i 端点的横坐标分类，并存入数组 $X[1..2n]$

步 2 $X[0]\leftarrow X[1]$，$m\leftarrow 0$(m 是并的长度)，$c\leftarrow 0$(c 是重叠线段的次数)

步 3 **for** $i=1$ **to** $2n$ **do**

步 4 **if** $c\neq 0$ **then** $m\leftarrow m+X[i]-X[i-1]$

步 5 **if** $X[i]$是左端点 **then** $c\leftarrow c+1$
else $c\leftarrow c-1$

显然，该算法的时间复杂性为 $O(n\log n)$。算法中行 4，区间$[X[i-1],X[i]]$长度是否

加入到 m 中依赖于 c 是否为 0，$c\neq0$ 时，则长度加入 m，$c\neq0$ 表示扫描线碰到一区间的左端点。该算法可以推广到二维，如图 6-4 所示。

图 6-4 中第 i 个垂直条是 $X[i-1]$ 和 $X[i]$ 之间的平面条，该平面条的有效面积(即图中阴影部分)是 $(X[i]-X[i-1])\times m_i$，其中 m_i 是条中一条任意垂直线和矩形并的截段的长度，用 $O(\log n)$ 时间可以确定 m_i，m_i 是计算矩形并的面积的重要参数。为了计算 m_i，把矩形的垂直边构造成线段树。定义结点 v 的参数 $m[v]=[B[v],E[v]]$ 提供给 m_i 的长度，计算 $m[v]$ 的过程如下：

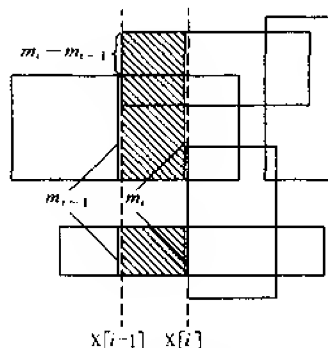


图 6-4　计算垂直边的长度

```
if  c[v]≠0  then  m[v]←E[v]-B[v]
else  if  v 不是叶
        then  m[v]←m[LSON[v]]+m[RSON[v]]
        else  m[v]←0
```

v 为线段树的根结点时，$m_i=m[v]$。当矩形的一条垂直边插入线段树或从线段树删去时，每个结点用常数时间能保持参数 $c[v]$ 和 $m[v]$。因此，矩形的每条垂直边用时间 $O(\log n)$ 可以插入线段树或从线段树中删去并计算 m_i。设横坐标 $X[i]$ 处垂直边的上端点、下端点的纵坐标分别为 e_i 和 b_i，则计算矩形并的面积的算法如下。

计算 F 的面积的算法

步 1　矩形垂直边按端点的 x 坐标分类，并存于数组 $X[1..2n]$；

步 2　$X[0]\leftarrow X[1]$，$m\leftarrow0$；

步 3　构造和初始化矩形边的纵坐标的线段树 T；

步 4　**for**　$i=1$　**to**　$2n$　**do**

```
    m'←m[root(T)];
    m←m+m'×(X[i]-X[i-1]);
    if  X[i]是左边的横坐标  then INSERT(b_i,e_i;root(T))
    else  DELETE(b_i,e_i;root(T))
```

步 1 耗费时间 $O(n\log n)$，其他步的时间限界均不超过此量级，因此计算 F 面积的算法的时间复杂性为 $O(n\log n)$。

步 1 中的数组 $X[1..2n]$ 提供事件点的进度表。

步 3 中 T 给出扫描线状态，它是由 $d-1$ 维推广到 d 维的关键，也是扫描方法的关键。事实上，扫描方法把给定的 d 维问题变为 n 个 $d-1$ 维子问题的序列。比如，$d=3$ 时，子问题变为二维，如果能用 $O(n\log n)$ 时间求解每个子问题，那么用 $O(n^2\log n)$ 时间可以求解原问题。利用四叉树和切片之间的粘合技术可以把此限界改进到 $O(n^2)$，此结果能否进一步改进，有待深入地研究。

为了计算矩形并 F 的周长，首先注意 m_i 的含义，m_i 是以 $X[i]$ 为右侧的垂直长条中垂直截线的总长度，同样 m_{i-1} 是以 $X[i-1]$ 为右侧的垂直长条中垂直截线的总长度，因此，m_i-m_{i-1} 是 $X[i-1]$ 处 F 的边界上垂直边的总长度。如图 6-4 所示，截线在区间 $[X[i-1],X[i]]$ 中是不变的，所以 F 的边界上垂直边的总长度在区间 $[X[i-1],X[i]]$

内不变，该总长度只是在扫描线通过点 $X[i]$ 处发生变化。其次要考虑垂直条 $[X[i-1],X[i]]$ 中水平边界边提供的长度。该长度由下式计算

$$(X[i]-X[i-1])\times\alpha_i$$

其中 α_i 是 $[X[i-1],X[i]]$ 条中 F 的边界上水平边数目，α_i 与 m_i 相似。图 6-4 中，$\alpha_i=4$，它必为偶数。对线段树的每个结点 v 定义 3 个参数 $\alpha[v]$、$LBD[v]$ 和 $RBD[v]$ 如下：

$\alpha[v]$ 是（F 的现时垂直截线）$\cap[B[v],E[v]]$ 的不相交部分数的两倍；

$$LBD[v]=\begin{cases}1 & B[v]\text{是（}F\text{ 的现时垂直截线）}\cap[B[v],E[v]]\text{中一个区间的下端；}\\0 & B[v]\text{不是（}F\text{ 的现时垂直截线）}\cap[B[v],E[v]]\text{中一个区间的下端。}\end{cases}$$

$$RBD[v]=\begin{cases}1 & E[v]\text{是（}F\text{ 的现时垂直截线）}\cap[B[v],E[v]]\text{中一个区间的上端；}\\0 & E[v]\text{不是（}F\text{ 的现时垂直截线）}\cap[B[v],E[v]]\text{中一个区间的上端。}\end{cases}$$

这 3 个参数的初值均为 0，下述过程计算该 3 个参数：

```
if  c[v]>0  then  α[v]←2;LBD[v]←1;RBD[v]←1
else  α[v]←α[LSON[v]]+α[RSON[v]]-2RBD[LSON[v]]LBD[RSON[v]];
      LBD[v]←LBD[LSON[v]];
      RBD[v]←RBD[RSON[v]]
```

$c[v]>0$ 时，区间 $[B[v],E[v]]\subset F$ 的现时垂直截线，所以 $\alpha[v]=2$，$LBD[v]=RBD[v]=1$；而 $c[v]=0$，并且 F 的现时垂直截线不包含跨接点 $E[\text{LSON}[v]]=B[\text{RSON}[v]]$ 的区间时，（F 的现时垂直截线）$\cap[B[v],E[v]]$ 包含的项和它的两个子结点中的项的和一样多，用 $RBD[\text{LSON}[v]]=LBD[\text{RSON}[v]]=1$ 来描述该情况。因此该计算过程是正确的。在每个结点处，用常数时间计算参数 α，LBD 和 RBD。

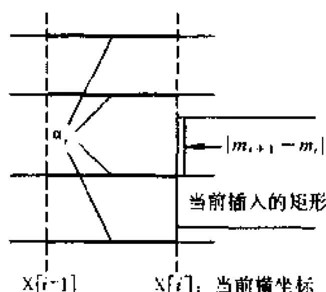


图 6-5　当前步提供给周长的长度由水平边和垂直边组成

修改 F 的面积的算法可以得到计算 F 周长的算法。当前步（垂直条）提供给周长的长度由两部分组成，如图 6-5 所示，在条 $[X[i-1],X[i]]$ 中的水平边，提供 $\alpha_i\times(X[i]-X[i-1])$；在横坐标 $X[i]$ 处的垂直边，提供 $|m_{i+1}-m_i|$。这样，α_i 的值要在后者修改之前从线段树中得到，$\alpha_i=\alpha[\text{root}(T)]$，而 m_{i+1} 是在修改之后得到的。

计算 F 的周长的算法

步 1　矩形的垂直边按端点的 x 坐标分类，并存于数组 $X[1..2n]$；

步 2　$X[0]\leftarrow X[1]$，$m_0\leftarrow 0$，$p\leftarrow 0$；

步 3　构造和初始化矩形边的纵坐标的线段树 T；

步 4　**for** $i=1$ **to** $2n$ **do**

$\alpha^*\leftarrow\alpha[\text{root}(T)]$；

if $X[i]$ 是左边横坐标

then $\text{INSERT}(b_i,e_i;\text{root}(T))$

```
else  DELETE(b_i, e_i; root(T));
m* ← m[root(T)];
p ← p + α* · (X[i] − X[i−1]) + |m* − m_0|;
m_0 ← m*
```
该算法用 $O(n\log n)$ 时间可以计算 F 的周长。

6.4 矩形并的轮廓

给定平面上 n 个同等安置矩形 $R_1, R_2, \cdots, R_n$，它们的并的边界称为矩形并的轮廓。并的轮廓包含若干个不相交的圈，而圈由交替的垂直边和水平边组成。仍沿用前面的规定，外轮廓方向为逆时针方向，此时矩形并在外轮廓的左边；内轮廓（即洞的边界）方向为顺时针方向，矩形并仍在内轮廓的左边。本节介绍寻找矩形并的外轮廓、内轮廓的算法。

6.3 节中阐述了计算 F 的周长的算法，F 的周长就是轮廓的长度，计算 F 的周长比求轮廓容易得多。求轮廓的一种算法，其基本思想是，先找出轮廓的垂直边的集合 V，然后用水平边连接垂直边便构成并轮廓的定向圈。在图 6-6 中有 5 个矩形，8 个事件点：x_1；x_2，x_3；x_4，x_5，x_6；x_7，x_8，x_9；x_{10}，x_{11}；x_{12}；x_{13}；x_{14}。各事件点所包含的元素个数不同，每个元素代表轮廓的一条垂直边，比如，第 2 个事件点包含两个元素 x_2，x_3，它们分别表示轮廓的两条垂直边 e_2，e_3。e_2 与 e_3 位于同一条扫描线上，并且 e_2 与 e_3 是不相交的两条线段。当扫描达到第 3 个事件点时，要修改并的截线，得到 3 条垂直边 e_4，e_5 与 e_6。每条垂直边 e_i 的方向不是向上就是向下，该方向的确定依赖于它位于矩形的右垂直边或左垂直边上。

得到垂直边集合 V 之后，用下述方法求得水平边：垂直边 e_i 表示为 $e_i=(x_i; b_i, t_i)$，其中 x_i 为横坐标，b_i 与 t_i 是纵坐标，并且 $b_i<t_i$。对每条垂直边 e_i 可以产生一对三元组 $(x_i, b_i; t_i)$ 与 $(x_i, t_i; b_i)$，每个三元组表示一点，一对三元组对应于 e_i 的两个端点，每个三元组的第 3 项是 e_i 的端点的一个基准。例如，图 6-6 中 e_2 表示为 $(x_2, b_2; t_2)$ 与 $(x_2, t_2; b_2)$，其中前者为 e_2 的下端点，而后者为 e_2 的上端点。以递增序字典式地分类三元组集合（先按纵坐标分类，然后再按横坐标分类）。图 6-6 中的水平边经分类后可表示为下列序列：

$y_1(x_2, b_2; t_2)(x_4, b_4; t_4)$，$y_2(x_7, b_7; t_7)(x_{10}, b_{10}; t_{10})$，$y_3(x_4, t_4; b_4)(x_7, t_7; b_7)$，
$y_4(x_{12}, b_{12}; t_{12})(x_{14}, b_{14}; t_{14})$，$y_5(x_5, b_5; t_5)(x_8, b_8; t_8)$，$y_6(x_1, b_1; t_1)(x_2, t_2; b_2)$，
$y_6(x_5, t_5; b_5)(x_8, t_8; b_8)$，$y_6(x_{10}, t_{10}; b_{10})(x_{12}, t_{12}; b_{12})$，$y_7(x_{13}, b_{13}; t_{13})(x_{14}, t_{14}; b_{14})$，
$y_8(x_1, t_1; b_1)(x_3, b_3; t_3)$，$y_8(x_6, b_6; t_6)(x_9, b_9; t_9)$，$y_8(x_{11}, b_{11}; t_{11})(x_{13}, t_{13}; b_{13})$，
$y_9(x_9, t_9; b_9)(x_{11}, t_{11}; b_{11})$，$y_{10}(x_3, t_3; b_3)(x_6, t_6; b_6)$。

该序列的元素个数为偶数，通过对该序列的一次扫描，便可产生所有水平边并找出与它们相连的垂直边，这样便给出轮廓的圈。用圈的最小横坐标的垂直边 e_j 来确定每个圈，当 e_j 是向上边时，该圈为洞的边界，而当 e_j 是向下边时，圈是轮廓的外边界。

线段树支持的平面扫描可以生成垂直轮廓边的集合，引入结点参数 STATUS[v] 定义如下：

$$\text{STATUS}[v]=\begin{cases}\text{满的，} & \text{若 } c[v]>0;\\ \text{部分的，} & \text{若 } c[v]=0\text{，但对 } v \text{ 的某个子孙 } u, c[u]>0;\\ \text{空的，} & \text{对于根在 } v \text{ 处的子树中的每个 } u, c[u]=0\text{。}\end{cases}$$

这样，当前的截线 G 是线段树中所有 STATUS[v]为满的结点上的所有线段[$B[v]$, $E[v]$]的并。给定矩形的一条垂直边 $s=(x;b,e)$，$s\cap\overline{G}$ 是区间[b,e]中 G 的相继截段之间空隙的序列，如图 6-7 所示，其中 $\overline{G}$ 是 G 的补。

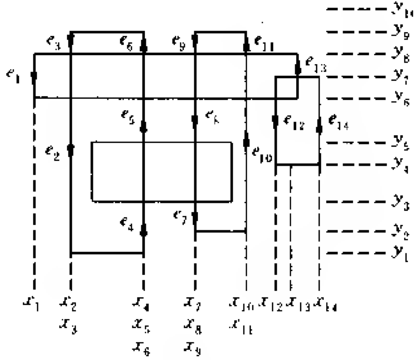


图 6-6 矩形并 F 的轮廓的一个实例

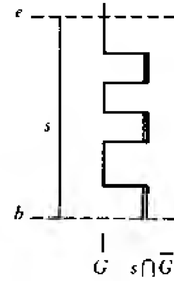


图 6-7 $s\cap\overline{G}$ 的序列

定义结点 v 的参数 Contr(v)如下：

$$
\text{Contr}(v)=[B[v],E[v]]\cap\overline{G}
=\begin{cases}
\phi & \text{若 STATUS}[v]\text{是满的，或对 } T \text{ 中 } v \text{ 的某个祖先 } u\text{，STATUS}[v]\text{是满的；}\\
[B[v],E[v]] & \text{若 STATUS}[v]\text{是空的；}\\
\text{Contr}(\text{LSON}[v])\cup\text{Contr}(\text{RSON}[v]) & \text{若 STATUS}[v]\text{是部分的。}
\end{cases}
$$

该表达式给出对应于 s 的一个截段的结点 v 提供给 $s\cap\overline{G}$ 的份额 Contr(v)，因此仅当 STATUS[v]是部分时，要查找根在 v 处的子树。下面是 Lipski 和 Preparata 提出的计算矩形并的轮廓的算法，它报告了集合[b,e]$\cap\overline{G}$ 以及公共横坐标。

计算矩形并的轮廓的算法(L-P 算法)

步 1 将垂直边的横坐标分类，并存入 $X[1..2n]$。

步 2 $A\leftarrow\phi$/A 是垂直边的集合/

步 3 构造和初始化矩形的纵坐标的线段树 T；

步 4 **for** $i=1$ **to** $2n$ **do**

```
        if  X[i]是左侧边的横坐标
          then  A←Contr(b_i,e_i;root(T))∪A;
                INSERT(b_i,e_i;root(T))
          else  DELETE(b_i,e_i;root(T));
                A←Contr(b_i,e_i;root(T))∪A
```

步 4 中垂直边集合的修改在左侧边插入之前，并且它要跟在右侧边删去之后，这就是说，如果一条右侧边和一条左侧边具有共同的横坐标，那么一定要在右侧边处理之前处理左侧边，步 1 满足这样的条件。另外还要设计计算 Contr 的子算法。Contr 子算法中使用了一个堆栈 $STACK$，开始时 $STACK$ 为空。这个子算法把线段序列[b,e]$\cap$Contr(v)推进堆栈。通过调用 Contr(b,e;root(T))返回堆栈的内容。

```
function  Contr(b,e;v)
    if  STATUS[v]≠满的  then
        if  b≤B[v]∧E[v]≤e∧STATUS[v]=空的
          then  /提供[B[v],E[v]]/
            if  B[v]=top(STACK)  then  /合并邻接的线段/
                    删去 top(STACK)
            else  STACK⇐B[v]/边的开始/;
                  STACK⇐E[v]/边的现时终止/
        else  if  b< ⌊(B[v]+E[v])/2⌋
              then  Contr(b,e;LSON[v]);
              if  ⌊(B[v]+E[v])/2⌋<e
              then  Contr(b,e;RSON[v])
```

可以证明,该算法的时间复杂性是 $O\left(n\log n+p\log\left(\frac{n^2}{p}\right)\right)$,其中 n 为同等安置矩形的数目,p 是矩形并的轮廓的边数目。这个问题的一个已知下界是 $\Omega(n\log n+p)$,而通过把分类问题变换为寻找矩形(无洞)并的轮廓得到一个 $\Omega(n\log n)$ 下界,因此该算法不是最优的。

计算 F 的轮廓的另一种方法是采用 6.1 节中平面扫描方法,先求出矩形边之间的交点,然后再求轮廓。

计算矩形并的轮廓的算法($Z_{6\text{-}1}$)

输入 n 个同等安置的矩形 R_i,每个矩形 R_i 的顶点 p_{ij} 按逆时针方向排列,$i=\overline{1,n}$,$j=\overline{1,4}$.

输出 矩形并的轮廓。

步 1 用 6.1 节中算法求出矩形边之间的交点 q_{ij},交点存于 A 中,并记录相交的边。

步 2 以横坐标值最小的边(垂直边)为起始边,沿逆时针方向行进,碰到交点 q,改沿另一矩形边并按逆时针方向行进,$A\leftarrow A-\{q\}$,直至回到出发边,便得到外轮廓。

步 3 **if** $A=\phi$ **then** 终止(输出轮廓)

else 从任一交点出发,沿矩形边按逆时针方向行进,碰到交点 q,改沿另一矩形边并按逆时针方向行进,$A\leftarrow A-\{q\}$,直至回到出发点,便得到一个内轮廓。重复步 3,直至 $A=\phi$。

步 2 和步 3 中,"碰到交点 q",其中交点 q 是指离当前点距离最近并在行进边上的点。

6.5 矩形并的闭包

在某些应用中,需要将矩形并加以扩充才能获得问题的圆满解决,这就导致矩形并的闭包概念。如果点 p_1,p_2 的 x 坐标和 y 坐标满足关系式 $p_2.x\leq p_1.x$,$p_2.y\leq p_1.y$,则称点 p_1 优于点 p_2,记为 $p_2<p_1$。关系"<"称为优势关系。给定平面中两点 $p_1=(x_1,y_1)$和 $p_2=$

(x_2,y_2)，如果 p_1 和 p_2 不满足优势关系，即关系式 $(x_1-x_2)(y_1-y_2)<0$ 成立，那么称点 p_1 和 p_2 是不可比的。而点 $p_1'=(x_1,y_2)$，$p_2'=(x_2,y_1)$ 分别称为 p_1 和 p_2 的西南(SW)共轭和东北(NE)共轭，如图 6-8 所示。

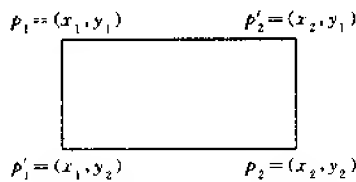


图 6-8　p_1 与 p_2 是不可比的，它们的共轭是 p_1' 和 p_2'

给定平面上区域 R 及 R 内任意两点 p_1、p_2，若 R 内存在连接 p_1 与 p_2 的一条曲线，那么称 R 中 p_1 与 p_2 是连通的。若对于 R 中任何两个连通的不可比点 p_1 和 p_2，p_1 和 p_2 的东北(西南)共轭也在 R 中，则称 R 是东北(西南)闭的。定义平面区域 S 的东北闭包(西南闭包)是包含 S 的最小东北(西南)闭区域 R，记为 NE(S)(SW(S))。如果包含 S 的最小区域 R，既是西南闭的又是东北闭的，那么称为 S 的东北西南闭包(简称闭包)，记为 NESW(S)。

给定平面曲线 L 上任意两点 $p_1=(x_1,y_1)$，$p_2=(x_2,y_2)$，如果由 $x_2>x_1$ 可以推得 $y_2\geqslant y_1$，则称曲线 L 对 x 单调。如果 L_1 与 L_2 是两条平面曲线，D 是一个禁止区域，一条曲线能连续地变换成另一条曲线，而且不与 D 相交，则称 L_1 与 L_2 是同伦的。如用 $R(F)$ 表示最小同等安置的矩形，而且它包围一个给定(不一定连通)的平面域 F，则有关系式

$$F\subseteq \mathrm{NESW}(F)\subseteq R(F)$$

若 NESW(F)是连通的，则 $R(F)$ 的东北角和西南角属于 F 的闭包，如图 6-9(a)所示。对于 NESW(F)的每个连通部分 G(图 6-9(b)所示)，NESW(F)的边界由两条 x 单调的曲线组成，分别同伦于 $R(G)$ 的上左和下右边界部分。这两条曲线的每一条是它的同伦曲线簇中不和 G 相交的 x 单调曲线的末端，称这两条曲线为闭区域 NESW(F)的上轮廓和下轮廓。对于任意域 F(不连通的)，如果 F 的闭包 NESW(F)由多个部分组成，那么任意两个部分之间一定是由一条 x 单调的曲线分隔开，如图 6-9(c)所示。

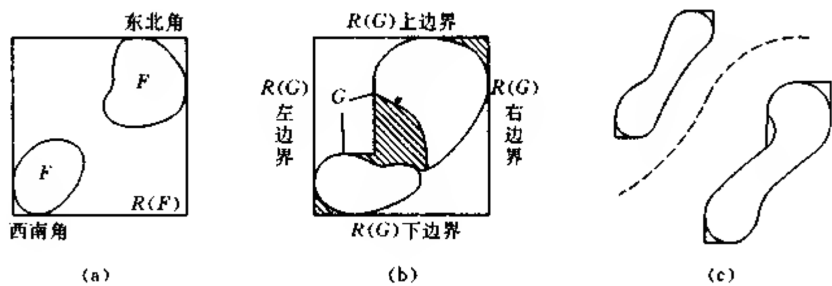


图　6-9

(a) 平面域 F 和它的最小包围的矩形 $R(F)$；

(b) F 的 NESW 闭包的一个连通部分；

(c) NESW(F)的两个部分的可分性

另外，下面的等式成立

$$\mathrm{NESW}(F)=\mathrm{NE}(\mathrm{SW}(F))=\mathrm{SW}(\mathrm{NE}(F)) \tag{6-1}$$

平面上给定 n 个同等安置的矩形，求它们的并的闭包。对于这个问题，依据式(6-1)可以设计两个算法用来计算域 F 的闭包，即先计算 F 的东北闭包，然后计算 NE(F)的西南闭包，或者先计算 F 的西南闭包，然后计算 SW(F)的东北闭包。如果 F 是同等安置矩形的并，那么计算工作就简单了，因为 NESW(F)的任何部分的上轮廓和下轮廓都是对 x 单调的。

计算 F 的闭包的算法

步 1 计算 n 个同等安置矩形 $R_1, R_2, \cdots, R_n$ 的并 F。

步 2 用从左到右扫描的方法构造 F 的东北闭包(确定东北闭的组成部分)NE(F)。

步 3 用从右到左扫描的方法构造 NE(F)的西南闭包。

在平面扫描中，与扫描线相交的 NE(F)的组成部分称为活动的。NE(F)的部分是活动的当且仅当该部分至少包含与扫描线相交的 F 的一个矩形(称为活动的矩形)。

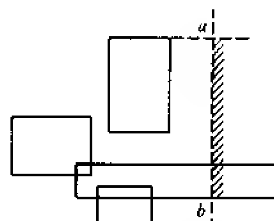


图 6-10 活动部分和它的活动区间

考虑步 2，显然，事件点是矩形垂直边的横坐标。当扫描线碰到一个矩形的左侧边时，便起始一个活动的部分，或扩大到一个活动部分，或合并多个活动部分；当扫描线碰到矩形右侧边时，结束一个活动部分。扫描线上活动区间的上端点 a 是至今所找到的所有矩形水平边的最大纵坐标，而它的下端点 b 是组成部分中现时活动矩形水平边的最小纵坐标。图 6-10 中阴影条表示活动区间。显然，步 2 构造了 F 的东北闭包。

当扫描线碰到左侧边时可能出现三种情况，利用活动区间可以区分这三种情况，如图 6-11 所示，图 6-11(a)表示起始一个新的部分；图 6-11(b)为扩大一个已存在的部分；图 6-11(c)是跨接两个部分。

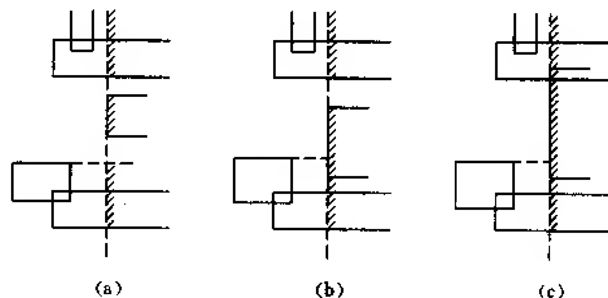


图 6-11 扫描线碰到左侧边时可能出现的三种情况

图 6-12 示出扫描线碰到矩形 R 的右侧边时可能出现的三种情况：图 6-12(a)表示 R 成为不影响活动区间的不活动矩形；图 6-12(b)向上收缩活动区间使矩形 R 成为不活动矩形；图 6-12(c)组成部分被终止。其中仅当 R 是部分的最底下的活动矩形时才出现图 6-12(b)情况；当 R 是组成部分仅有的活动矩形时出现图 6-12(c)情况。

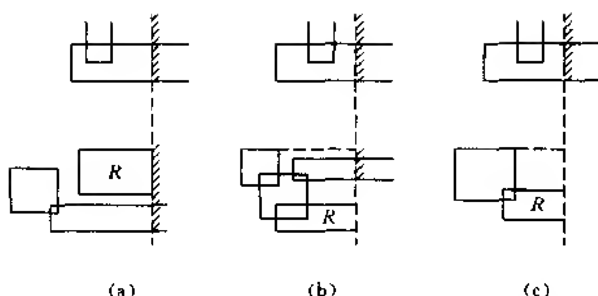


图 6-12 扫描线碰到右侧边时可能出现的三种情况

利用高度均衡线段树 T 的叶子存储活动区间的端点，当处理左侧边$[y_1,y_2]$时，先在 T 中定位 y_1，然后遍历叶子，直至 y_2 被定位。用 $O(\log n+h)$时间完成此操作，其中 h 是合并的活动区间的个数(图 6-11(c))。每个矩形只有一条左侧边，所以插入工作的上界为 $O(n\log n)$。扫描线碰到右侧边$[y_1,y_2]$时，从 T 中删去该边，耗费时间 $O(\log n)$。

用自右向左扫描方法构造 SW(NE(F))的过程与上述相似。矩形边的横坐标的分类耗费 $O(n\log n)$时间，处理 $2n$ 条边的耗费也是 $O(n\log n)$时间，因此，用时间 $\theta(n\log n)$能构造 n 个同等安置矩形的并的 NESW 闭包。

6.6 矩形并的非平凡轮廓和外轮廓

6.4 节中介绍了矩形并的轮廓及计算轮廓的两种算法。n 个矩形的并的轮廓可以有 $\theta(n^2)$条边，但对于轮廓的一部分，比如外轮廓，则只有 $O(n)$条边。矩形并 F 的外轮廓是 F 和平面的无界域之间的边界。除了外轮廓和内轮廓之外，还有非平凡轮廓。同等安置矩形的并 F 的非平凡轮廓是轮廓圈的集合，并且它们的每一个至少包含给定矩形的一个顶点，因此有关系式

$$轮廓 = 外轮廓 \cup 内轮廓$$
$$外轮廓 \subseteq 非平凡轮廓$$
$$非平凡轮廓 - 外轮廓 \subseteq 内轮廓$$

不包含矩形顶点的轮廓称为平凡轮廓，如图 6-13 所示，其中图 6-13(a)表示 F 的轮廓，图 6-13(b)为非平凡轮廓，图 6-13(c)是平凡轮廓，图 6-13(d)为外轮廓。

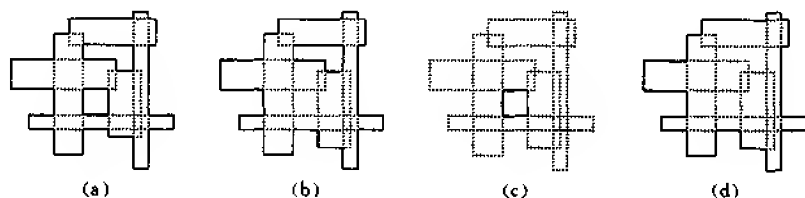


图 6-13 矩形并 F 的轮廓

每个矩形的边把平面划分为两个区域，即矩形内部区域和外部区域，其中外部区域是

无界的。对于无界的外部区域，规定其边界为顺时针回路，而对于矩形并来说，它亦称为外部回路。规定内部区域的边界为逆时针回路，又称为内部回路。若回路的一条弧包含矩形边的一个端点，则称该弧是末端的。如果一条回路至少包含一条末端弧，则称此回路是非平凡的，否则为平凡回路。图 6-14 示出矩形并的非平凡回路及外部回路，显然，外部回路是一种特殊的非平凡回路，即外部回路＝外轮廓⊆非平凡轮廓⊆非平凡回路。

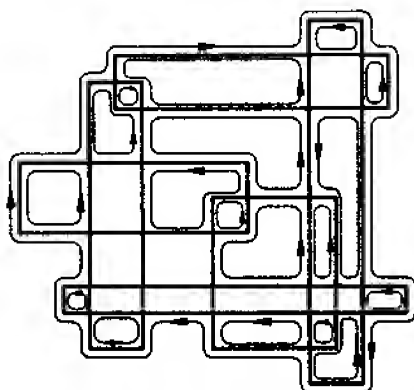


图 6-14　矩形并的非平凡回路及外部回路

设平面上给定 n 个同等安置矩形 $R_1,R_2,\cdots,R_n$，求它们的并的非平凡轮廓和外轮廓。可以证明，n 个同等安置矩形并的非平凡回路的弧的数目为 $O(n)$。Lipski 和 Preparata (1981)提出了一种求解该问题的算法，其基本思想是，沿着构造的轮廓的回路执行一次"行进"，每次加入一条弧，每加入一条弧耗费 $O(\log n)$时间，因此总的时间是 $O(n\log n)$。行进是一个前进过程，即从当前矩形顶点 v_1 开始，按规定的方向沿当前线段 l_1 行进，如图 6-15所示，出现两种情况之一：

(1) 行进碰到与 l_1 相交的线段 l_2(离 v_1 最近)，交点 v_2，l_2 穿过矩形域到 l_1 的左侧。此时，行进向左转，交点 v_2 变成当前顶点，l_2 为当前线段，如图 6-15(a)所示。

(2) 达到 l_1 的端点 v_3，v_3 也是 l_2 的端点。此时，行进向右转，v_3 变成当前顶点，l_2 为当前线段，如图 6-15(b)所示。

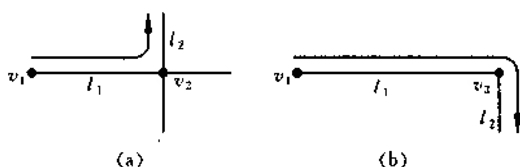


图 6-15　*L-P* 算法中行进过程可能出现的情况

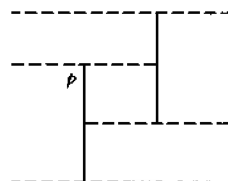


图 6-16　水平邻接图

利用查找水平邻接图和垂直邻接图的方法可以实现上述的行进。矩形的垂直边构成集合 V，显然 $|V|=2n$。过垂直边的端点 p 向左、向右作水平半直线，这些水平半直线终止于最接近于 p 的垂直边上；或者没有这样的垂直边时，便延长水平半直线到无穷。经过这样处理之后，平面被划分为若干个区域，称为广义矩形，又称为水平邻接图，如图 6-16 所

示。用归纳法可以证明，水平邻接图中区域总数至多为 $3|V|+1$。

设当前(矩形的)顶点为 v_1(见图 6-15)，从 v_1 出发的 4 个方向上的线段为当前线段，现只考虑向右的水平方向上的线段 l_1(其他情况类似)，设 $v_1=(x_1,y_1)$，l_1 的另一端点的坐标为(x_3,y_1)。在水平邻接图中先定位点(x_1,y_1)，然后求得 v_1 右侧的最接近垂直线段的横坐标 x_2，若 $x_2\leqslant x_3$，则行进向左转(图 6-15(a))；若 $x_2>x_3$，则行进向右转(图 6-15(b))。因此，把一条弧加入到一个非平凡回路要耗费邻接图的一次询问(即一个平面点定位)，该询问时间为 $O(\log n)$。非平凡回路中有 $O(n)$条弧，用 $O(\log n)$时间可以完成一条弧的构造，所以构造非平凡回路集合的时间为$O(n\log n)$。

在获得非平凡回路之后，删去位于矩形内部的非平凡回路(其弧位于矩形内部)，便得到非平凡轮廓，再删去逆时针方向回路，即得到外部轮廓。用 $\theta(n\log n)$时间可以构造 n 个同等安置矩形并的非平凡轮廓和外轮廓。

6.7 矩形的交

本节讨论同等安置矩形的交。显然，只有当两个同等安置矩形至少有一个公共点时，它们才相交。先考虑一维情况，设 $A=[a_1,a_2]$和 $B=[a_1',a_2']$是两个区间，条件 $A\cap B\neq\phi$ 等价于下列互斥的条件之一：

$$a_1\leqslant a_1'\leqslant a_2 \tag{6-2}$$

$$a_1'<a_1\leqslant a_2' \tag{6-3}$$

易见，$(a_1\leqslant a_1'\leqslant a_2)\vee(a_1'<a_1\leqslant a_2')$等价于 $a_1'\leqslant a_2\wedge a_1\leqslant a_2'$，即$-a_2\leqslant -a_1'\wedge a_1\leqslant a_2'$，后者是平面中点$(-a_1',a_2')$和点$(-a_2,a_1)$之间的一个优势关系$\prec$，也就是$(-a_2,a_1)\prec(-a_1',a_2')$。这表明只要把区间变换为平面中的点，便可以把二维问题(确定 n 个同等安置矩形集 Q 中的所有相交的矩形对)与一维问题(确定 n 个区间的所有相交的区间对)联系起来。

为了解决二维问题，采用平面扫描方法，事件点仍取为矩形的垂直边的横坐标。由矩形和扫描线的交给出扫描线状态，与扫描线相交的矩形仍称为活动矩形。在图 6-17 中，当前事件点是矩形 $R=[x_1,x_2]\times[a_1',a_2']$的左侧边，扫描线横坐标 x_1(也是 R 的 x 区间的左端)属于每个活动矩形的 x 区间。因此，只需决定哪个活动区间关于 y 区间条件(6-2)成立，或条件(6-3)成立。利用对线段树的查找可以报告相交的矩形对，也就是说，当插入矩形的左侧边$[a_1',a_2']$时，通过检测活动区间$[a_1,a_2]$，是否$a_1'\leqslant a_1\leqslant a_2'$，或者是否 $a_1\leqslant a_1'\leqslant a_2$ 来完成。当找到相交的线段对时，便找到了相交的矩形对。产生事件点列耗费时间 $O(n\log n)$，用时间 $O(\log n)$插入或删去每个区间，树的查找用时间 $O(\log n)$来跟踪查找路径，因此确定 n 个同等安置矩形的 s 个相交对耗费 $\theta(n\log n+s)$时间。

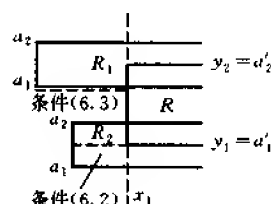


图 6-17 扫描线碰到 R 的左侧边时，出现的活动矩形 R_1 和 R_2

下面介绍解决二维问题的另一种方法。

给定 n 个区间的集合$\{R^{(j)}=[a_1^{(j)},a_2^{(j)}],j=\overline{1,n}\}$，首先定义函数 $\sigma_0:\{R^{(1)},R^{(2)},\cdots,$

$R^{(n)}\} \to E^2$，它把区间$[a_1,a_2]$映射到平面中的点(a_1,a_2)，该点位于第Ⅰ象限的平分线的上方。其次引入函数$\sigma_1: E^2 \to E^2$，它是平面上关于x_2轴的对称映射，即点(a_1,a_2)映射到点$(-a_1,a_2)$。再引入函数$\sigma_2: E^2 \to E^2$，它是平面上关于第Ⅱ象限的平分线$x_2=-x_1$的对称映射，即点$(-a_1,a_2)$映射到点$(-a_2,a_1)$，如图6-18所示。为方便起见，记复合函数$f_1=\sigma_1\sigma_0$，$f_2=\sigma_2\sigma_1\sigma_0$，并且有等价关系：

$$R^{(i)} \cap R^{(j)} \neq \phi \Leftrightarrow f_1(R^{(i)}) > f_2(R^{(j)}) \Leftrightarrow f_1(R^{(j)}) > f_2(R^{(i)}) \tag{6-4}$$

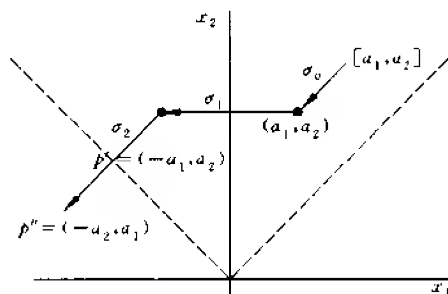


图6-18 f_1，f_2将区间映射到两个点p'和p''

图6-19是这种等价关系的一个例子。显然，相交的区间对引出优势关系中的两个对。

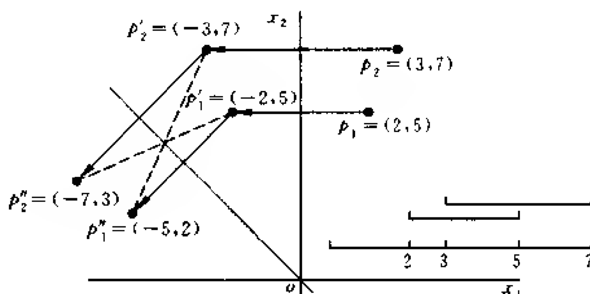


图6-19 等价关系的例子

给定矩形$R=[x_1,x_2]\times[y_1,y_2]$，先把$R$映射到四维空间$E^4$中的两个点$q'(R)=(-x_1,x_2,-y_1,y_2)$和$q''(R)=(-x_2,x_1,-y_2,y_1)$，其次形成$E^4$中的两个点集：$S'=\{q'(R)|R\in Q\}$，$S''=\{q''(R)|R\in Q\}$。由式(6-4)，得：$R_1$、$R_2\in Q$，$R_1\cap R_2\neq\varphi$当且仅当$q'(R_1)>q''(R_2)$，或者$q'(R_2)>q''(R_1)$。因此，确定所有相交对问题是优势问题（给定$E^4$中两个点集$S'$和$S''$，找所有对$q'\in S'$，$q''\in S''$，使得$q'>q''$）的一个特例，而四维优势问题存在复杂性为$O(n\log^2 n+s)$的算法求解它，因此求解所有相交对问题的时间复杂性是$O(n\log^2 n+s)$。

另外，两个矩形的位置关系有4种基本情况，如图6-20所示。图中阴影部分是它们的交，每种基本情况产生的交都不相同。图6-20(a)所示的交，两个交点位于交域的对角线上；图6-20(b)中的交，两个交点位于一个矩形的右侧边上；图6-20(d)的交，两个交点位于一个矩形的左侧边上；图6-20(c)中的交域由4个交点组成。多个矩形的交均由上述4

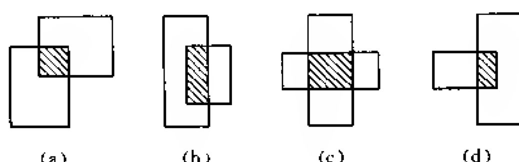


图 6-20 两个矩形之间的位置关系

种基本情况组合而成，为了求出多个矩形的所有交，采用平面扫描方法找出交点并区分各种基本情况，便可找出所有交。

求矩形交的算法($Z_{6\text{-}2}$)

步 1 按矩形垂直边的 x 坐标分类。

步 2 扫描线从左向右移动，碰到左侧边时，检查是否有水平边落入该左侧边区间内。如果有，则求出交点。

若有 1 个交点，扫描线右移求出另外 1 个交点，便找到 1 个交域。如图 6-20(a)所示情况。

若有 2 个交点，扫描线右移求出另外 2 个交点，便找到 1 个交域。如图 6-20(c)所示。

若有 2 个交点，扫描线右移碰到的矩形右侧边位于另一活动矩形内部，便找到 1 个交域。如图 6-20(d)所示。

步 3 碰到的左侧边位于另一活动矩形内部，扫描线右移求出 2 个交点，产生图 6-20(b)所示的交。一个矩形的左、右侧边位于另一活动矩形内部，则两个矩形呈包含关系。

6.8 应用举例

本节介绍矩形并的闭包的一个应用实例，即数据库中的并发控制。考虑多个用户并发存取到一个数据库的调度问题，每个用户通过锁定 x(数据库的变量)，修改 x 和开锁 x 等步骤系列实现事务处理。我们的目的是要开发一个事务处理系统以解决上述的调度问题。如果有两个用户、一个变量 x，那么可以抽象如下：用二维笛卡儿空间 E^2 的坐标轴来确定每个用户对变量 x 的处理过程，具体办法是建立事务处理步和坐标轴上正整数坐标点之间的一一对应关系，即坐标轴上的正整数点表示锁定 x 或开锁 x 的时刻，而锁定 x 和开锁 x 之间的区间表示修改 x 的时间间隔，也就是锁定区间。图 6-21(a)表明了这种情况，平面上的点代表数据库的状态，比如点(7,2)表示当前执行用户 U_1 的第 7 步和用户 U_2 的第 2 步，图中的矩形域是 U_1 和 U_2 对变量 x 的锁定域。显然，事务处理系统中状态点不能落入关联于变量的任何矩形域的内部。

如果有两个用户和 3 个变量 x_1、x_2 与 x_3，每个用户可存取 $x_i(i=\overline{1,3})$，变量 x_i 的每次存取由涉及“锁定 x_i”、“修改 x_i”和“开锁 x_i”的子序列完成。图 6-21(b)表明了这种情况，坐标轴上的点集表示一个用户的步序列(用户的活动)。我们用进度表来表示数据库活动的演变。在图 6-21(b)中，曲线 b、c 均为进度表，它们是单调不减的阶梯曲线，每条阶梯曲

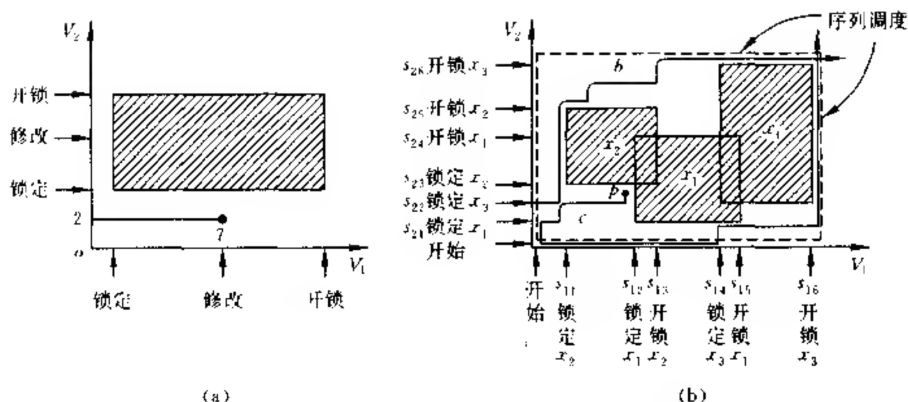


图 6-21　两个用户并发存取到一个数据库的几何模型

线代表用户对变量 $x_i(i=\overline{1,3})$的处理过程，比如曲线 c=开始 $s_{21}s_{11}s_{22}s_{12}s_{23}$描述的过程：开始；用户 U_2 在时刻 s_{21}锁定 x_1；用户 U_1 在时刻 s_{11}锁定 x_2；用户 U_2 在时刻 s_{22}锁定 x_3；用户 U_1 在时刻 s_{12}锁定 x_1；用户 U_2 在时刻 s_{23}锁定 x_2。这就是说，曲线 c 达到点 p 时，3 个变量均被锁定，两个用户都不可能进行下一步处理。显然，代表进度表的曲线不可能穿过事务处理矩形的并 F，再加上曲线 c 所代表的情况要加以排除，因此，只有位于 F 的西南闭包的外部，并且是单调非降的曲线所代表的进度表才能保证处理过程是无死锁的。例如图 6-21(b)中曲线 b 所表示的进度表是无死锁的处理过程，利用 6.5 节中求矩形并 F 的西南闭包算法先求出 F 的西南闭包，然后再求类似于 b 的曲线，便可解决这里的问题。

第 7 章　几何体的排列

平面上的直线或三维空间中平面的排列是 S 计算几何中的重要结构，它与凸壳、Voronoi 图等具有同样的重要性。本章将介绍有关排列的一些基本概念、算法和应用。

7.1　基本概念

设平面上给定 n 条直线 $s_1, s_2, \cdots, s_n$，这些直线将平面划分成凸域（或称网格或称面）、线段（两个交点之间）以及顶点（两条直线的交点），称这种划分为直线的排列。图 7-1 示出 8 条直线的一种排列。

在直线的排列中，区域（或面）称为排列的 2-面。直线 s_i 被其他直线分成 n 个区间，这些区间称为排列的边或者排列的 1-面。所有直线的交点称为排列的顶点或者排列的 0-面。我们通常把排列的 2-面看成开域（不包括它们的边），而将 1-面看成是开线段（不包括端点）。显然，j-面的维数是 j。这样，n 条直线的排列覆盖了整个平面，但它的 2-面与 1-面及 0-面之间不相交，不同的 j-面（j=0，或 1，或 2）之间也不相交。

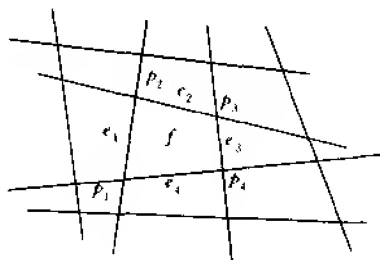


图 7-1　8 条直线的一种排列

直线的排列推广到 d 维空间 E^d 中是超平面的排列，每个超平面是满足 $a_1x_1+\cdots+a_dx_d=a_0$ 形式的线性等式的 E^d 中点的集合，其中 x_i 表示 E^d 中的坐标，系数 a_i 是任意实数，并且 $a_1, \cdots, a_d$ 不全为 0。当 $d=2$ 时，超平面是一条直线，而 $d=3$ 时，超平面是一个平面。

在 E^d 中给定超平面集合 N，由 N 构造的排列 $A(N)$ 是划分 E^d 为具近邻关系且变化维的域（面）。也就是说，N 中超平面将 E^d 划分成若干个凸域，这些 d-维凸域称为排列 $A(N)$ 的网格或者 d-面。在每个确定的超平面 $s_i \in N$ 上，N 中其他超平面与 s_i 的交产生 s_i 范围内的一个 $(d-1)$-维排列，该 $(d-1)$-维排列的网格叫做 $A(N)$ 的 $(d-1)$-面。用同样的方法继续下去，对所有的 $j<d$，定义 $A(N)$ 的 j 面。$A(N)$ 的 0-面也称为顶点，1-面叫做边。

如果 N 中（$d=2$）每一对直线交于一个点，则称排列 $A(N)$ 是简单的，这意味着 N 中没有 3 条直线交于一点，并且没有两条直线是平行的。从某种意义来说，非简单的排列是退化的。

n 条直线组成的集合 N 上，所有简单排列 $A(N)$ 具有相同的顶点数、边数及面数。

定理 7-1　n 条直线的简单排列中，顶点数 $V=\binom{n}{2}$，边数 $E=n^2$，面数 $F=\binom{n}{2}+n+1$。并且任何非简单排列的顶点数、边数及面数都不可能超过这些数量。

证明 由 n 条直线简单排列的定义可以推得 $V=\binom{n}{2}$。利用关于直线条数的简单数学归纳法可以证明 E 的公式。当 $n=2$ 时，两条直线的交点将两条直线分割成 4 段，即 $E=2^2$，故结论成立。假设 $n-1$ 条直线的任意简单排列 $A(N-1)$ 有 $(n-1)^2$ 条边，插入一条新的直线 s_n 到 $A(n-1)$，S_n 把 $A(n-1)$ 中 $n-1$ 条直线的每条直线上的一条边分成两段，同时 s_n 本身被 $A(n-1)$ 中的 $n-1$ 条直线分成 n 条边，因此，当 $A(n)$ 中有 n 条直线时，$E=(n-1)^2+(n-1)+n=n^2$。故 E 的公式成立。

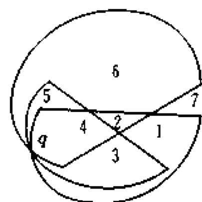


图 7-2　改造直线的排列成为带有相同参数(边、面)的平面图，这里 $n=3$ 并有 7 个面

为了证明 F 的公式，先将 n 条直线的排列 $A(n)$ 改造成与 $A(n)$ 具有相同参数(边数和面数)的图 G：首先，在 $A(n)$ 中 n 条直线上截取 n 条线段(保留所有交点)，并在 $A(n)$ 的无穷域中任取一点 q，然后将各线段的端点与 q 连接起来，如图 7-2 所示。这样便得到平面图 G，G 有 $V+1$ 个顶点、E 条边和 F 个面，其中 V、E 和 F 分别是 $A(n)$ 的顶点数、边数和面数。注意，这时有一个面(图 7-2 中编号 7 的面)成为 G 的外部面。依据欧拉公式 $V-E+F=2$ 便可以得到 $(V+1)-E+F=2$，或者 $F=E-V+1$，再代入 V 和 E 的已知值，便推得

$$F = n^2 - \frac{n(n-1)}{2} + 1 = (n^2+n+2)/2 = \binom{n}{2} + n + 1$$

对于非简单排列，至少有 3 条直线交于 1 点，所以 $A(n)$ 的顶点数、边数及面数均不会超过上述表达式所表示的限界。　　证毕。

当有 $k>2$ 条直线交于 1 点时，可以稍微移动(干扰)这些直线，使其没有 3 条及 3 条以上直线交于 1 点，如图 7-3(a)所示。如果 $A(N)$ 中有两条直线平行，我们只要稍微转动其中的 1 条，就可以改变原来的非简单排列成为简单排列，如图 7-3(b)所示。显然，在这两种情况下，$A(N)$ 的参数 V、E 和 F 的值都将增加。这说明，非简单排列转变成简单排列将增加排列的组合复杂性，因此非简单排列不可能是最坏的情况。

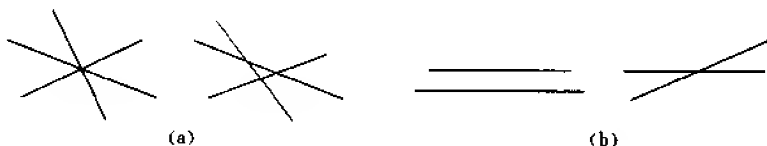


图 7-3　干扰非简单排列中的直线，使其成为简单排列

依据定理 7-1，V、E 和 F 的数量都是 $\theta(n^2)$，所以构造平面上直线排列的算法均具有二次方复杂性。

当 $d=2$ 时，n 条直线的简单排列 $A(N)$ 可以表示为一个平面图。另外，可以用面网格结构表示 $A(N)$，它是一个树结构，其结点代表一个 j-面，并且每个结点包含辅助信息，比如指向包含对应面的超平面的指针。如果 j-面 f 与 $(j-1)$-面 g 相邻，即 g 是 f 的边界，那么对应于 j-面 f 的结点与对应于 $(j-1)$-面 g 的结点构成父子关系。图 7-4 是图 7-1 中关于排列的面网格结构的一部分。

为了设计有效的构造排列的算法，要介绍排列的一个重要组合性质，即在已有的排列

中增加一条直线，该直线所能穿过的网格边的数目不会过多，这就是下面阐述的区段定理。

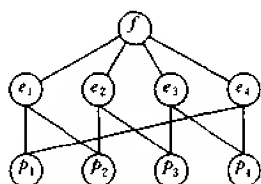


图 7-4　图 7-1 中排列的部分面网格结构

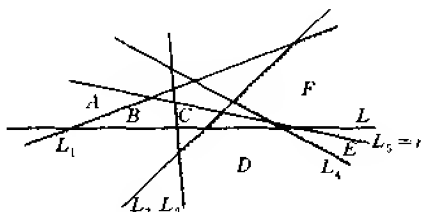


图 7-5　$A(N)$中 L 的区段

假设已给定 n 条直线的简单排列 $A(N)$及直线 L，又设 $A(N)\cup\{L\}$是简单排列。用 $Z_{A(N)}(L)$或 $Z(L)$表示 $A(N)$中 L 的区段，显然，$Z(L)$是 $A(N)$中与 L 相交的网格(面)的集合。比如，在图 7-5 中 $Z(L)=\{A,B,C,D,E,F\}$。区段定理限界这些网格(面)的边的总数，该总数用 $z(L)$表示，令 $|A|$是网格(面)A 的边的数目。图 7-5 中，$|A|=2$，$|B|=4$，$|C|=3$，$|D|=4$，$|E|=2$ 及 $|F|=4$。注意，相邻网格(面)的共同边界计数两次。图 7-5 中，$z(L)=|A|+|B|+|C|+|D|+|E|+|F|=19$。另设 $z_n=\max\limits_{L}(z(L))$，$z(L)$是 n 的函数。z_n 可作为 7.2 节中增量算法的复杂性的限界。

定理 7-2　设 $A(N)$是 n 条直线的排列，$L\in A(N)$，$A(N)$中与 L 相交的网格(面)的边的总数是 $O(n)$，更确切地是 $z_n\leqslant 6n$。

证明　不失一般性，假设简单排列 $A(N)$中插入一条新直线之后仍是简单排列，并且寻找区段的直线 L 是水平的，以及假设 $A(N)$中没有垂直直线。由于简单排列的复杂性是最坏的，所以不考虑退化情况的上述假设是合理的。

因为没有直线是垂直的，故划分 $Z(L)$的每个网格(面)的边成左边界和右边界是可行的，图 7-5 中用点线表示网格(面)的左边界。网格(面)C 只有一条左边界，该左边界的上、下端点是网格(面)C 的最高、最低顶点，而且是唯一的。网格(面)B、D 与 E 分别有唯一的最高顶点，但都没有最低顶点。网格(面)F 既没有最高顶点，也没有最低顶点。另外，左边界和右边界有对称作用，因此仅需考虑对 z_n 产生影响的左边界的数目，并记左边界数目为 l_n，$l_n\leqslant 3n$。图 7-5 中，$n=5$，$l_n=9$。

采用对 n 的归纳法证明。$n=0$ 时，空排列无左边界，所以 $l_0\leqslant 0$ 成立。假设 $l_{n-1}\leqslant 3n-3$成立。设 $A(N)$是满足假设的 n 条直线的排列，选择与直线 L 交于最右边的一条直线，如图 7-5 中的 L_5，记 L_5 为 r。从 $A(N)$中删去 r，得到 $A(N-1)$，它是 $A(N)-\{r\}$的排列，并且有 $n-1$ 条直线，因而归纳假设成立，即 $l_{n-1}\leqslant 3n-3$。现在把 r 插回到 $A(N-1)$，此时可以使 l_{n-1}至多增加 3，下面证明这一点。$r=L_5$ 插入 $A(N-1)$之后，将原来的一条左边界(图 7-5 中 L_4)分成两段，再加上新插入的 r，同时其他某些网格(面)也将发生变化。

从图 7-5 中删去 r 之后形成的排列 $A(N-1)$，如图 7-6 所示，图中网格(面)A'，B'，C'，D'和 G'等均与直线 L 相交。在 $A(N-1)$中插入 r 之后(见图 7-5)，A'，B'，C'与 D'均被割去部分子域，G'被划分成 E 和 F。B'变成 B 之后，左边界仍是 1 条边，D 的左边界数目仍是 2，C 的左边界变成 1 条边(减少 1)，E 和 F 的左边界数目比 G'的左边界数目增加 2。

总之，插入 r 后，左边界数目由 8 增加到 9。

当插入新的直线时，仅需要左边界数目增长的上界，而不是精确的数值。另外，选择最右边直线进行插入是为了得到左边界的限界。

选择 r 可以得到与 $A(N)$ 中的 L 最右的交，记该交为 $x=r\cap L$。x 必然位于 $A(N-1)$ 中与 L 相交的最右网格(面)中，即图 7-6 中的 G'。r 上的一部分将成为 $A(N)$ 的最右边网格的左边界，因为 x 在 r 上，并且由 x 向右的射线必定在最右的网格中。因此，r 至少包含一个新的左边界。但 r 不包含 $A(N)$ 中任何其他的左边界(r 不包含图 7-5 中几条新的左边界)，由于包含不止一条左边界的 $A(N)$(比如图 7-5 中 L_3)的任意直线 α 必定被一条分裂网格的直线 β (比如 L_4)切割，但另一方面，β 在 α 的右边与 L 相交。这样，α 与 L 的交不可能位于最右边，因此选择了 r。

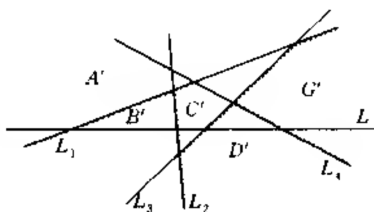


图 7-6 排列 $A(N-1)=A(N)-\{r\}$

得到 r 只包含 1 条新的左边界的结论之后，仅需限制 r 分裂原有左边界成两段，比如图 7-5 中的 $r=L_5$ 将图 7-6 中 G' 的 1 条左边界分裂成两段，其中一段是 F 的左边界，另一段是 E 的左边界。这样分裂只能发生在 $A(N-1)$ 中 L 上的最右网格内(图 7-6 中 G' 内)，因为 r 是割去而不是分裂与它相交的所有其他网格。

这样，r 仅能分裂最右网格的左边界。因为网格是凸的，r 至多只能与它的两条左边界相交，所以 r 至多可以分裂两条原有的左边界。因此，插入 r 之后可以增加 1 条新的左边界，并且至多分裂两条原有的左边界，而每分裂 1 条原有的左边界便增加 1 条左边界，使得 l_{n-1} 至多增加 3，得到 $l_n\leqslant 3(n-1)+3=3n$。 证毕。

7.2 确定直线排列的算法

为了设计构造直线排列的算法，首先要解决算法的输入、输出的表示问题。可以用直线的斜率和截距来表示输入直线，用类似于多面体面的表示方法或图 7-4 中的表示方法表示直线排列的输出。

下面介绍构造直线排列的增量算法。假设已构造 $i-1$ 条直线的排列 $A(i-1)$，插入第 i 条直线 L_i 之后，要求构造排列 $A(i)$，为此，需要求出 $A(i-1)$ 与 L_i 的所有交点。我们首先用常数时间找到 L_i 与 $A(i-1)$ 中的任意一条直线的交点 x，如图 7-7 中，$x=L_i\cap L_2$。然后由 x 向前沿 L_i 的区段 $Z(L_i)$ 的每个面的边按顺时针方向前进，即图 7-7 中用曲线表示的路径。直至遇到 L_i 与另一条直线相交并求出交点，重复此过程，当遇到区段的一条无穷边时前进终止。图 7-7 中，从 x 出发经 C 的三条边遇到 L_3 和 L_i 相交并求出交点之后，重复该过程，即经过

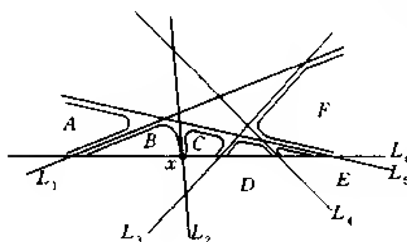


图 7-7 直线 L_i 插入排列

D 的三条边，E 的两条边和 F 的三条边，最后进入 F 的一条无穷左边界。这样便求出 L_i 与 L_3、L_4、L_5 的交点，终止该过程。同理，从 x 向后沿 B 的边界按逆时针方向经过 B 的三条边，求出 L_1 与 L_i 的交点之后，再沿 A 的边界按逆时针方向经过 A 的一条边，最后进入 A 的一条无穷右边界，终止该过程。经过每个面的边界耗费常数时间，插入一条直线的总耗费取决于区段的复杂性，由定理 7-2 知，该复杂性是 $O(n)$。注意，如何使用排列的结构来避免分类，在找到与 L_i 相交的所有交点之后，耗费 $O(n)$ 时间可以将 $A(i-1)$ 修改为 $A(i)$。因此整个构造过程需要 $O(n^2)$ 时间。

构造直线排列的增量算法

步 1 构造 $A(0)$，空排列的一个数据结构。

步 2 **for** $i=1,2,\cdots,n$ **do**

将直线 L_i 插入 $A(i-1)$，过程如下：

寻找 L_i 和 $A(i-1)$ 中某直线的交点 x。

由 x 出发沿 $Z(L_i)$ 中网格向前移动。

由 x 出发沿 $Z(L_i)$ 中网格向后移动。

修改 $A(i-1)$ 成为 $A(i)$

定理 7-3 平面上 n 条直线的排列可以在 $\theta(n^2)$ 时间及空间内构造。

证明 上面的分析已证明该算法需要 $O(n^2)$ 时间。正如定理 7-2 中所见到的那样，结构也是这么大，因此，在最坏情况下存储该结构需要 $O(n^2)$ 空间。 证毕。

上述关于直线排列的概念、理论与算法可以推广到高维。比如，d 维超平面的排列中任意维的面的数目是 $O(n^d)$，任意超平面的区段的复杂性为 $O(n^{d-1})$，在 $O(n^d)$ 时间和空间内可以构造这样一种排列。具体地说，$d=3$ 时，三维中平面的排列具有复杂性 $O(n^3)$，并且可以在 $O(n^3)$ 时间内构造。

7.3 对偶性

S 计算几何中，对偶变换将起重要的作用，因为它们可以把一个问题变换成另一个问题，从而使问题的求解获得新的途径。本节介绍三种对偶变换并说明其相关的性质。

首先考虑凸多面体与凸壳。给定 E^d 中点的集合 S，定义 S 的凸壳 $\mathrm{CH}(S)$ 是包含 S 中所有点的最小凸集。当 S 是点集时，$\mathrm{CH}(S)$ 表示 S 的凸壳；而当 S 是半空间集合时，$\mathrm{CH}(S)$ 表示由 S 形成的凸多面体。如果点 $p(\in S)$ 是 $\mathrm{CH}(S)$ 的一个顶点，则称点 p 是 S 中的一个极值点，否则称为非极值点。E^d 中凸壳的简单例子是 d-单纯形，而组成它的每个侧面是 $(d-1)$-单纯形。2-单纯形恰好是一个三角形，1-单纯形是一条边。如果 $\mathrm{CH}(S)$ 的所有 j-面是维数 j 的单纯形，则称 $\mathrm{CH}(S)$ 是单纯的。图 7-8(a) 中的 $\mathrm{CH}(S)$ 是单纯的，而图 7-8(b) 中的 $\mathrm{CH}(S)$ 不是单纯的，因为它的底面不是单纯形。

给定 E^d 中点的集合 $\hat{S}$，定义 $\hat{S}$ 的上凸壳是集合 $\hat{S}\cup\{(0,\cdots,0,\infty)\}$ 的凸壳。

凸多面体和凸壳是对偶几何结构，也就是说，存在一个几何变换将凸多面体变换成凸壳，反之亦成立。考虑映射点 $p=(a_1,\cdots,a_d)\in E^d$ 到超平面 $\langle p,x\rangle=a_1x_1+\cdots+a_dx_d=1$ 的变换 T，其中 $\langle p,x\rangle$ 表示向量的内积，并且假设 $a_1,\cdots,a_d$ 不全为 0。T 的逆变换亦成立。显

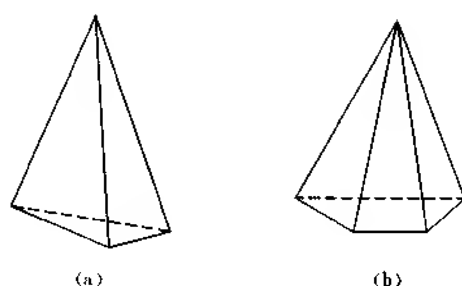


图 7-8
(a) CH(S)是单纯形的 (b) CH(S)不是单纯形的

然有 $T(T(p))=p$。令 CH(S)是由 E^d 中半空间集合 S 构成的任意 d-多面体，并设 $\hat{S}$ 表示由变换 S 中超平面所得到的 E^d 中点的集合，如图 7-9 所示。

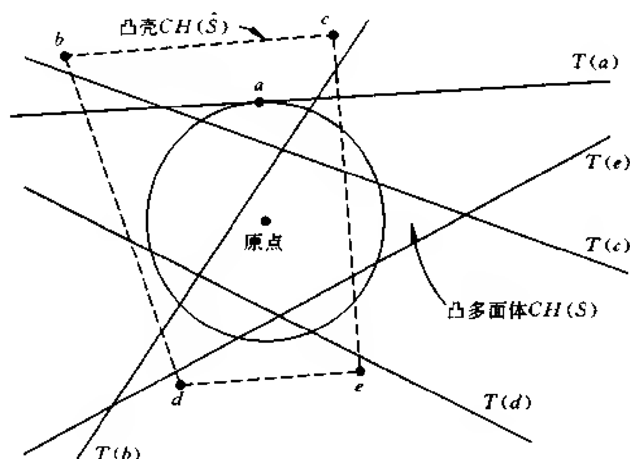


图 7-9 对偶变换 T
——：S 中的线 •：$\hat{S}$ 中的点

变换 T 将单位球 $x_1^2+\cdots+x_d^2=1$ 上的点 p 映射到在 p 处与单位球正切的超平面。一般说来，它映射离原点 o 距离为 d 的点 p 到离原点距离为 $\frac{1}{d}$ 的超平面，该超平面垂直于射线 $\overrightarrow{op}$。

对于 E^d 中的超平面 h，令 h^- 表示包含原点的超平面所限界的半空间，h^+ 为另一半空间。如果用系数向量 $(h_1,\cdots,h_d)$ 及方程 $h_1x_1+\cdots+h_dx_d=1$ 表示超平面 h，那么对于任意点 $p=(p_1,\cdots,p_d)$，

$$\langle p,h\rangle = \langle T(p),T(h)\rangle \tag{7-1}$$

此外，点 p 位于超平面 h 上，当且仅当 $\langle p,h\rangle=1$；点 p 位于 h^+ 中，当且仅当 $\langle p,h\rangle\leqslant 1$。依据式(7-1)，变换 T 有下述性质：

(1) 关联不变性：点 p 位于超平面 h 上，当且仅当点 $T(h)$ 位于超平面 $T(p)$ 上。

(2) 包含不变性：点 p 位于半空间 $h^+(h^-)$，当且仅当 $T(h)$ 位于 $T(p)^+(T(p)^-)$。

这些性质表明(见图 7-9)：

(1) 超平面 h 变换到 CH(S)内一点，该超平面的限界半空间 h^+ 的内部严格地包含 $\hat{S}$ 的点。

(2) 更为一般地，超平面 h 变换到面 CH(S) $\bigcap_{p\in \bar{J}} T(p)$ 的内部的一个点，该超平面包含点的子集 $J\subset\hat{S}$，并且它的限界半空间 h^+ 内部严格地包含 $\hat{S}$ 的剩余点。

这意味着：

(1) S 中冗余超平面变换到 $\hat{S}$ 中非极值点，反之也成立。

(2) S 中非冗余的超平面变换到 $\hat{S}$ 中极值点，反之也成立。这样 CH(S)的面与CH($\hat{S}$)的顶点一一对应。

(3) 更一般地，对于 $0\leqslant j\leqslant d-1$、CH($S$)的 j-面与 CH($\hat{S}$)的 $(d-j-1)$-面之间存在一一对应关系。CH(S)包含于 S 中超平面 $S_1,S_2,\cdots$ 内，而 CH($\hat{S}$)是点 $\hat{S}_1,\hat{S}_2,\cdots$ 的凸壳。

(4) 如果 CH(S)是单纯的，则 CH($\hat{S}$)是单纯形的，反之亦成立。

类似地，上凸多面体和上凸壳是对偶几何结构，它们之间的对偶变换是用抛物面 $x_d=x_1^2+\cdots+x_{d-1}^2$ 并让$(0,\cdots,0,\infty)$起原点的作用代替变换 T 中的单位球所得到的，该变换称为对偶变换 D。

现令 S 是 E^d 中超平面集合，对任意超平面 $h\in S$，h^+ 表示由 h 开始沿 x_d 轴正向延伸产生的半空间，而 h^- 表示另一半空间。考虑将每个点$(p_1,\cdots,p_d)$映射到超平面 $x_d=2p_1x_1+\cdots+2p_{d-1}x_{d-1}-p_d$ 的变换 D，反之亦成立。变换 D 把位于抛物面 $x_d=x_1^2+\cdots+x_{d-1}^2$ 上的点 p 映射到在 p 处与抛物面正切的超平面。此外，如果两点 p_1 和 p_2 位于平行于 x_d 轴的直线上，则已变换的超平面是平行的并且它们之间有相同的垂直距离，如图 7-10 所示。

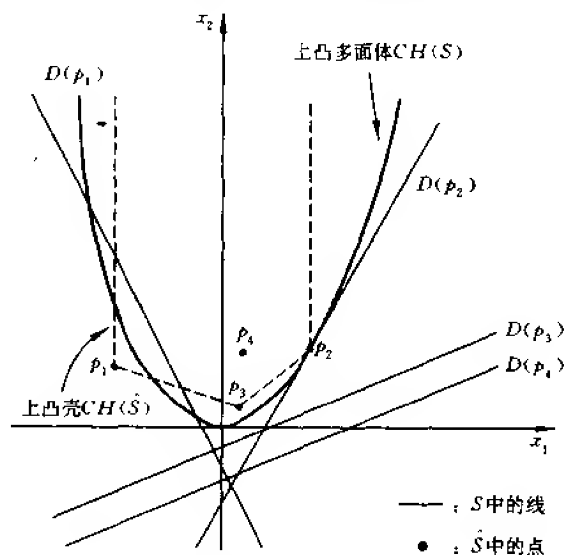


图 7-10 对偶变换 D

变换 D 具有下述性质：

(1) 关联不变性：点 p 位于超平面 h 上，当且仅当点 $D(h)$ 位于超平面 $D(p)$。

(2) 包含不变性：点 p 位于半空间 $h^+(h^-)$，当且仅当点 $D(h)$ 位于 $D(p)^+(D(D)^-)$。

设 $\hat{S}$ 是 S 中通过变换超平面得到的点集。上述性质表示 S 的上凸多面体和 $\hat{S}$ 的上凸壳是对偶的几何体，如图 7-10 所示。CH(S)的 j-面和 CH($\hat{S}$)的($d-j-1$)-面之间存在一一对应关系，对于 $0\leqslant j\leqslant d-1$，也存在这种对应关系，其中 CH($S$)的 j-面包含于 S 中超平面 $S_1, S_2, \cdots$ 的交内，而 CH($\hat{S}$)的($d-j-1$)-面是点 $\hat{S}_1, \hat{S}_2, \cdots$ 的凸壳。

下面再介绍点与直线之间的一种对偶变换 D'。把平面上点 p 的坐标 x、y 分别作为直线的斜率和截距，这就建立了平面上点与直线之间的变换 D'，即直线 $L: y=mx+b$ 对应于点 $p=(m,b)$。为了方便起见，我们取 $L: y=2ax-b$ 对应于点 $p=(a,b)$。显然有 $D'(L)=p$ 并且 $D'(p)=L$，但上述对应关系并不明显，如果选取 $b=a^2$，则直线 $y=2ax-a^2$ 在点(a,a^2)处与抛物线 $y=x^2$ 相切。这样，变换 D' 把点 $p=(a,b)$变换为切线。如果 $b<a^2$，则 $D'(p)$把点 p 映射到平行于切线但高于切线的一条直线；若 $b>a^2$，则 $D'(p)$把点 p 映射到切线下面的一条平行线，如图 7-11 所示。

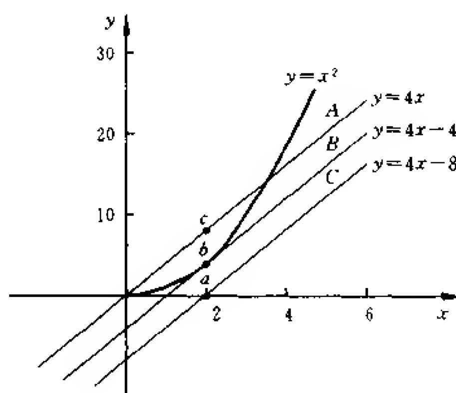


图 7-11 $D'(a)=A, D'(b)=B, D'(c)=C$

对偶变换 D' 具有下述基本性质：

(1) D' 是其自身的逆，即 $D'(D'(x))=x$，其中 x 是一个点或是一条直线。

(2) D' 是平面上所有非垂直线与所有点之间的一一对应关系。

(3) 点 p 位于直线 L 上，当且仅当点 $D'(L)$位于直线 $D'(p)$上。

(4) 直线 L_1 和 L_2 交于点 p，当且仅当直线 $D'(p)$通过两点 $D'(L_1)$和 $D'(L_2)$。

(5) 如果点 p 位于直线 L 的上(下)方，则直线 $D'(p)$位于点 $D'(L)$的下(上)方。例如，在图 7-11 中，点 c 在直线 B 的上方，而直线 C 在点 b 的下方。

作为对偶性的一个应用，考虑半空间域的查询问题。设 S 是 E^d 中点的集合，给定半空间域 h，目的是报告或者计算落入 h 中的 S 的所有点。如果使用上面介绍的对偶变换 D 将 S 中的点映射到 E^d 中的超平面，并且类似地把限界 h 的超平面 $\bar{h}$ 映射到点 $D(\bar{h})$，那么半空间域的查询问题变换为下述问题：报告或者计算位于点 $D(\bar{h})$上面或者下面的 $D(S)$

$=\{D(p)\mid p\in S\}$中的所有超平面，它依赖于 h 位于 $\bar{h}$ 的下面还是位于 $\bar{h}$ 的上面，如图 7-12(a)、(b)中所示。如果在由 $D(S)$超平面形成的排列中确定任意 d-网格 Δ，那么点 $q\in\Delta$ 上面的(或者下面的)超平面的集合仍然是相同的。因此，半空间域查询问题和超平面排列中点定位问题构成了对偶关系。

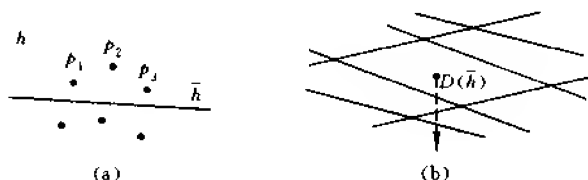


图 7-12

(a) 半空间域查询 (b) 对偶空间中半空间域查询

7.4 Voronoi 图

本节讨论排列与 Voronoi 图之间的关系。

7.4.1 一维情况

假设在 x 轴上给定点集 $S=\{x_1,x_2,\cdots,x_n\}$，将 $x_i(i=\overline{1,n})$向上投影到抛物线 $y=x^2$ 上，其投影点坐标为(x_i,x_i^2)。在点(x_i,x_i^2)处，抛物线的切线方程为 T_i : $y=2x_ix-x_i^2$，该切线是 $D'((x_i,x_i^2))$，即 $D'((x_i,x_i^2))$等于抛物线 $y=x^2$ 在点(x_i,x_i^2)处的切线，反之 $D'(T_i)$等于点(x_i,x_i^2)。抛物线上两条相邻切线 T_i 与 T_{i+1}之间交点的 x 坐标是它们生成点 x_i 和 x_{i+1}之间的中点，即(x_i,x_i^2)和(x_{i+1},x_{i+1}^2)处的切线的交点的 x 坐标为$\frac{1}{2}(x_i+x_{i+1})$。这些交点垂直投影到 x 轴上便产生 S 的一维 Voronoi 图。在图 7-13 中，$S=\{-15,-3,1,10,20\}$，作抛物线在$(-15,225)$，$(-3,9)$，$(1,1)$，$(10,100)$，$(20,400)$等点处的切线，相邻切线的交点在 x 轴上的投影点-9，-1，$5\frac{1}{2}$，15 恰好是点集 S 的 Voronoi 图，图中用小圆圈表示。

在图 7-13 中，抛物线完全位于切线(直线)排列的网格 C 内，网格 C 的边界顶点在 x 轴上的投影点恰好是(一维)点集 S 的 Voronoi 图。下面换一种方式来观察，由抛物线上点(b,b^2)向下作垂直线 $x=b$，碰到网格 C 的第一条边，并将其映射到 b 所在的 Voronoi 网格中(x 轴上的一条线段)。为了解释这个观察，设 T 是在点(a,a^2)处与抛物线相切的一条直线，T 的方程为 $y=2ax-a^2$。抛物线与 $x=b$ 上方的 T 之间的垂直距离 d 是 a 与 b 之间距离的平方。事实上，$d=b^2-(2ab-a^2)=(b-a)^2$，如图 7-14 所示，其中$(b-a)<1$，因此 $d<(b-a)$。这表明，由抛物线上点(b,b^2)向下作垂直线时，如果先遇到切线 T_i，后遇到切线 T_j，那么 T_i 距 b 上方的抛物线比 T_j 距 b 上方的抛物线近，因此 b 离 x_i 比离 x_j 更近。

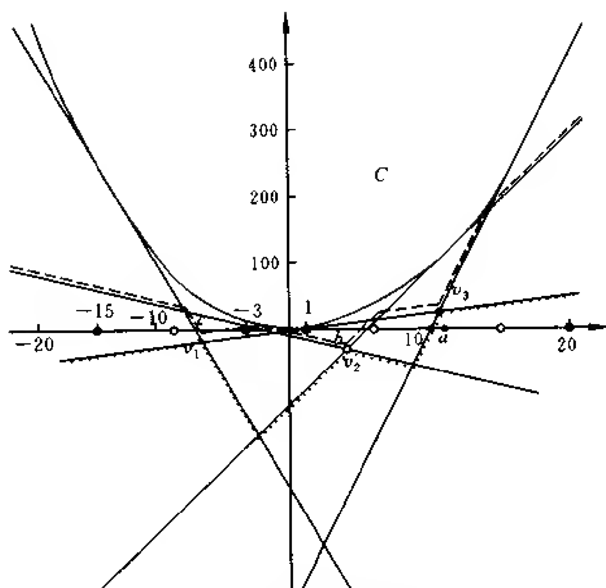


图 7-13　抛物线切线的排列

由上述观察可得到下面的结果。

定理 7-4　抛物线 $y=x^2$ 上由点 (b,b^2) 向下移动垂线 $x=b$，遇到切线的顺序与由 b 向左移动遇到的产生切线的 x_i 的顺序相同。

定理 7-4 表明切线的分类对应于最近邻近分类。

下面讨论排列与二阶 Voronoi 图的关系。定义二阶 Voronoi 图为划分有关空间(现只讨论 x 轴)成具有某种性质的点的域，该域中的点距 S 中点 p_i 距离最近，同时距 S 中另一点 p_j 距离次最近。在此定义中，p_i 与 p_j 的顺序无关紧要。因此，如果 a 的最近邻是 p_i 并且它的第二最近邻是 p_j，而 b 的最近邻是 p_j，第二最近邻是 p_i，那么 a 与 b 位于相同的二阶 Voronoi 域中。该二阶 Voronoi 图隐含在抛物线切线排列的边中，并且这些切线的垂直上方仅有另一条切线。这些边包括称为该排列的 2-层的那些边。图 7-13 中切线排列的 2-层边链用虚线表示。2-层边链的顶点在 x 轴上的投影点将 x 轴划分成若干个区间(退化的网格)，这些区间中的点 a 离 S 中点 p_i 最近，同时 a 离 S 中点 p_j 次最近。在图 7-13 中，当 $x>15$ 时，离 x 最近、次最近的 S 中的两个点是(20,10)；$10\frac{1}{2}<x<15$ 时，离 x 最近、次最近的两个点是(10,20)；$5\frac{1}{2}<x<10\frac{1}{2}$ 时，离 x 最近、次最近的两个点是(10,1)等等。

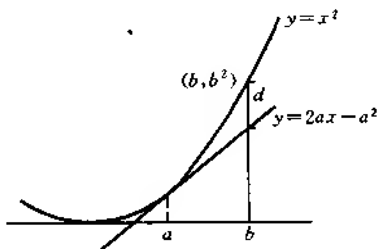


图 7-14　$d=(b-a)^2$

如果切线排列的边 e_i 的上方有 $i-1$ 条切线，则称 e_i 为排列的 i 层边，在图 7-13 中，排

列的 3-层边链用点线表示。图中 2-层边链(虚线)和 3-层边链交于三个点 v_1,v_2 与 v_3。设 v_3 在 x 轴上的垂直投影为点 a,过点 $x=a+\varepsilon(\varepsilon>0$ 且值很小)作垂线,沿此垂线从上到下将碰到三条切线,设为 A、B、C。显然,在 x 处,B 在 2-层上,C 在 3-层上。再观察点 $x=a-\varepsilon$ 处,C 在 2-层而 B 在 3-层,并且 B 和 C 的交点的 x 坐标为 a,所以 $x=a$ 表示从$\{A,B\}$到$\{A,C\}$的前两个最近邻近的变化,这证明 2-层边链和 3-层边链的交点 v_1、v_2 与 v_3 在 x 轴上的投影点是二阶 Voronoi 图中不同 Voronoi 域的分界点,也就是说,二阶 Voronoi 图中的边界点是排列的 2-层边链和 3-层边链交点的投影点。图 7-13 中,点 a,b,c 是 2 阶 Voronoi 图的边界点。2-层边链上的其他顶点(不在 3-层边链上)在 x 轴上的投影点表示两个最近邻近顺序的改变(比如,$x=15$,(20,10)的顺序变成(10,20)的顺序),但不改变最近邻近集合($x=15$ 处,最近邻近集合是$\{10,20\}$),所以 2-层边链的其他顶点(不在 3-层边链上)在 x 轴上的投影点不是二阶 Voronoi 图的边界点。这样便得到下面的结果。

定理 7-5 抛物线切线的排列中,i 层边链和 $i+1$ 层边链的交点在 x 轴上的投影点构成点集 S 的 i 阶 Voronoi 图。

7.4.2 二维情况

上述关于一维情况下的定义及结果可以推广到二维。给定平面上点的集合 $S=\{p_1, p_2,\cdots,p_n\}$,由 $p_i(i=\overline{1,n})$向上作垂线交抛物面于 q_i,过 q_i 作抛物面的切平面,这样便构成切平面的排列。该排列的 1 层面边界链的棱边在 xy 平面上的投影组成点集 S 的二维 Voronoi 图。同样,k 阶 Voronoi 图是 k 层和$(k+1)$层面边界链交的投影,它是边和顶点的集合。二阶 Voronoi 图的例子见图 4-42。因此,所有高阶 Voronoi 图嵌入正切平面的排列中。

由于三维中平面排列的构造需要 $O(n^3)$时间,而层都被嵌入排列中,层之间没有共同的面,即所有高阶 Voronoi 图嵌入正切平面的排列中,故构造所有 k 阶$(k=1,\cdots,n-1)$ Voronoi 图的复杂性是 $O(n^3)$。

7.5 应用

本节介绍排列的几种应用。

7.5.1 k-最近邻近

作为排列的一种间接应用,就是利用排列先构造 Voronoi 图,然后依据 Voronoi 图的定义,解决寻找查询点的最近邻近问题,而如果先构造 k 阶 Voronoi 图,那么寻找查询点的 k 个最近邻近问题也可以得到解决。这常用于"k 个最近邻近的判定规则":把未知量按它的 k 个最近邻近的表示进行分类。在设备定位、信息检索以及面插入等实际问题中,都可以碰到 k 个最近邻近问题。

7.5.2 删去隐藏面

现今人们利用计算机制作电视广告或者电影特技，在这些应用中最频繁使用的是删去隐藏面。例如将多种彩色多边形拼接成所需要的各种图案，如果这种拼接允许重叠，那么就要删去被遮盖的部分。

设输入多边形顶点数目为 n，并且一组多边形遮盖另一组多边形，如图 7-15 所示，这时至少产生$\frac{n^2}{16}$个交点，在删去由这些交点组成的某些多边形之后，才能显示出两组多边形重叠的场景。因此在最坏情况下最佳算法的时间复杂性不低于 $O(n^2)$。现已有处理该问题的许多算法，比如 $Z_{5\text{-}4}'$ 算法，它们的复杂性为 $O(n^2\log n)$，比该问题的复杂性下界高 $\log n$ 倍。McKenna 利用排列方法已设计出最坏情况复杂性为 $O(n^2)$的算法。下面简要介绍这种算法。

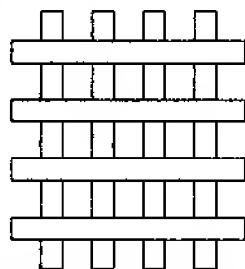


图 7-15 建立问题复杂性下界 $\Omega(n^2)$的实例

假设多边形在空间不相互穿透，即它们可以有重叠的部分和重叠的边界，但它们的内部是分离的；另外，假设视点离多边形无穷远，使得所有视线是平行的，并且不处理透视的情况。不失一般性，设眼睛在$(0,0,+\infty)$，即 z 轴正向无穷远处，这样，视野平面是 xy-平面，$z=0$。在所有多边形下面放置一个充分大的背景多边形 B，使得所有视线都碰到 B。问题是要确定由给定观察点可见到的场景。另外，一般问题可以变换到这个问题。

首先把输入凸多边形的每条边投影到 xy-平面（只要删去端点的 z 坐标），这称为正交投影。然后在 xy-平面上将投影线延伸成直线，这样便构成 xy-平面上 n 条直线的一种排列 $A(n)$，由定理 7-3 知，在$O(n^2)$时间内可以构造 $A(n)$。现要确定 $A(n)$的一个网格，它们是由多边形集合中位置最高的多边形投影到 xy-平面上形成的，该多边形离眼睛最近，而离 xy-平面最远。由于每个凸多边形投影到 xy-平面上只产生一个网格，所以可以对网格着与形成该网格的多边形相同的颜色。

一种简单的算法需要 $O(n^3)$时间，这是由于 $A(n)$有 $O(n^2)$个网格，对 $O(n^2)$个网格中的每个网格，计算 $O(n)$个多边形的每一个多边形的高度，而要求每个网格仅耗费常数时间。

Mckenna 算法使用了排列的拓扑扫描，他推广了 Edelsbrunner 和 Guibas 提出的平面扫描方法，这种方法不是扫描排列面上的一条垂直线，而是扫描一条垂直的虚设线 L，L 是与 $A(n)$中的每条直线仅相交一次的曲线，在交点处 L 从下向上穿过 $A(n)$中的直线。取 L 为曲线的优点是，在优先队列查找中不必耗费 $O(\log n)$时间确定哪个点是下一个要扫描的点。也就是说，可以保持可扫描顶点的一个无序的聚集：L 穿过的那些顶点与相邻的两条边关联。图 7-16 中顶点 v 是可扫描的，因为 L 穿过的网格 C 的两条相邻的边与 v 是关联的。

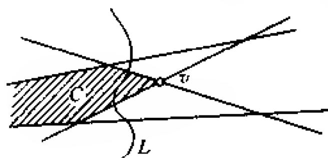


图 7-16 顶点 v 是可扫描的

除了确定的排列之外，算法保持的数据结构包括激活的网格表和 L 穿过的边（比如

图 7-16 中阴影网格)的表，并且对于每个激活的网格 C，其投影包含 xy-平面中网格 C 的所有多边形的表，将这些多边形按 z-深度进行分类。注意，只对激活的网格保持这些表。由于 L 穿过所有 n 条直线，所以总是有 $n+1$ 个网格。这些表能够提供足够的信息以确定 $A(n)$的每个网格的最前面的多边形。当扫描一个顶点时，旧的网格消亡而新的网格被激活，但它们的包含多边形的表或者相同或者几乎相同。为了使 $A(n)$的所有网格中每个网格的修改只耗费常数时间，可以利用这种相关性来传递穿过已扫描过的顶点的足够信息。这种删去隐藏面的算法在最坏情况下的复杂性是 $O(n^2)$。

实际应用时这个算法不是最好的，因为它总是需要 $\Omega(n^2)$时间和空间。在许多实际问题中，重叠部分没有图 7-15 那么复杂，因此要设计出对输出场景复杂性敏感的算法，这种算法称为输出规模敏感的隐藏面算法，这是当前的一个研究课题。

7.5.3 特征图

计算机视觉中的特征图(aspect graphs)的概念是为辅助图像识别提出的，其思想是存储一个物体可以提供给观察者的全部特有的视野，然后把这些视野与实际上所看到的视野进行比较。对于一个多面体，利用组合的等价性确定特有的视野：如果图像有相同的组合结构，也就是说，视野平面上多面体的可视面投影所产生的已标记的平面图是相同的，那么从两个视点所看到多面体的特征相同。可视空间划分(VSP)是一种把对象外部所有空间划分成连通域或者特征不变的网格。最后，特征图是 VSP 的对偶，每个域对应一个结点，并且给面的连通域分配一条弧。

排列为理解凸多面体的 VSP(或者对偶的特征图)提供了一种几何结构。对于多面体 P，其 VSP 的确是由包含 P 的面所在的平面形成的排列。例如，考虑图 7-17 所示立方体的排列，它划分空间成 26 个无界域，其中 6 个是基于立方体面的矩形柱体，8 个是与顶点关联的并位于顶角的柱体，以及 12 个楔，每个楔与立方体的一条边关联。考虑由网格 A 移动的点 p，从 p 观察立方体，穿过排列的面 f，进入相邻的网格 B，如图 7-17 中解释的那样。假设由网格 A 来观察，排列面 f 所位于的那个平面的立方体面 F 是可视的，那么当 p 在 f 上时，就是从边上看 F，而当 p 移动到网格 B 时，F 就不可视了。因此，f 的确表示特征的转变。

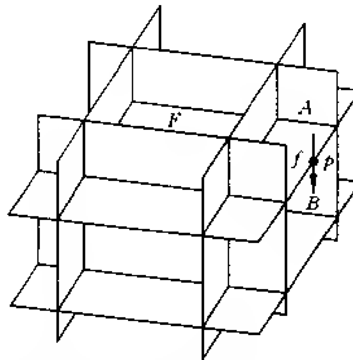


图 7-17　包含立方体面的平面排列，由网格 A 可视 F 而由网格 B 看不见 F

由定理 7-3 的推广能够得到 n 个顶点的凸多面体的 VSP 有规模 $O(n^3)$，并且可以在 $O(n^3)$时间内构造。依据 VSP 的表示，特征图是有效的。

7.5.4 点集的分割

给定平面上 n 个点的点集 S，并假设没有 3 个点在同一条直线上，要求作至少穿过一个点的平分线 L，L 将 S 分成 A 与 B，使得 $|A|=|B|$。

7.3 节中介绍了对偶变换 D'，D' 把平面上的点变换为直线，即 $p=(a,b)$ 对应于 $y=2ax-b$，这样，D' 使 S 中的点对偶化，产生 n 条直线的排列 $A(n)$。下面讨论点集 S 的所有平分线的确对偶于 $A(n)$ 的中间层 $M_{A(n)}$，$M_{A(n)}$ 是 $A(n)$ 的（以及它们连接的顶点）边的聚集，该边的点恰有 $(n-1)/2$ 条线分别位于其上方和下方。由对偶变换 D' 的基本性质(5)，点 $p\in M_{A(n)}$ 对偶于线 $D'(p)$，当 p 上方有线时，线 $D'(p)$ 的下方有相同数目的点。由中间层的含义，在 p 上方有 $(n-1)/2$ 条线，所以在 $D'(p)$ 下方有 S 的 $(n-1)/2$ 个点，也就是说，$D'(p)$ 平分 S，因此，$D'(p)$ 是一条平分线 iff $p\in M_{A(n)}$。

定理 7-6 点集的平分线对偶化为对偶排列线的中间层。

由该定理知，分割 A 和 B 的线必须对偶化为位于 $M_{A(n)}$ 和 $M_{B(n)}$ 上的点（其中 $B(n)$ 是对偶于 B 的排列）。因此，通过求两个集合中间层的交可以找到所有的分割。

设 A' 和 B' 是两个用一条直线可以分割开的点集，利用适当的平移和旋转，能够把它们变换到由 y-轴分割的集合 A 和 B（A 右，B 左），如图 7-18(a)所示。现把对偶变换 D' 应用于 A 和 B，排列 $A(n)$ 中的线的斜率均为正值，而 $B(n)$ 中的线的斜率均为负值，如图 7-18(b)，(c)所示。

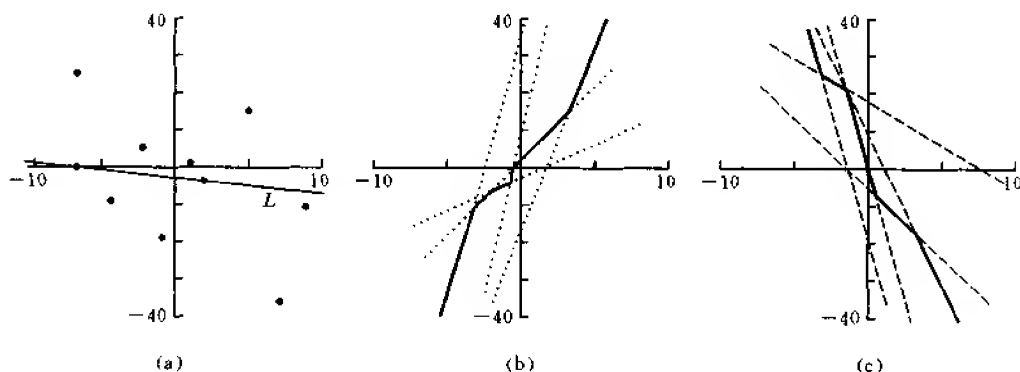


图 7-18

(a) 两个集合 A 和 B 中均有 5 个点，所示线 L 平分 A 和 B

(b) $x>0$ 时，点的对偶（(a)中点集 A）线有正斜率

(c) $x<0$ 时，点的对偶（(a)中点集 B）线有负斜率

由于 $M_{A(n)}$ 是由正斜率线的子线段组成，并且是严格单调增加的，同样，$M_{B(n)}$ 是严格单调减的（图 7-18(b)，(c)中用实线表示 $M_{A(n)}$ 和 $M_{B(n)}$）。因此 $M_{A(n)}$ 和 $M_{B(n)}$ 必交于一点，该交点的对偶线分割点集 A 和 B。图 7-19 中的交点 $\left(-\frac{1}{6},\frac{7}{3}\right)$ 的对偶线 L : $y=$

$2\left(-\frac{1}{6}\right)x-\frac{7}{3}=-\frac{1}{3}x-\frac{7}{3}$分割 A 和 B。

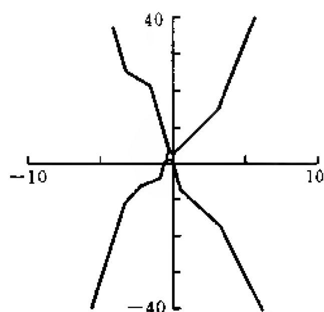


图 7-19　$M_{A(n)}$与 $M_{B(n)}$的交点

Edelsbrunner 提出的一个算法，对于有 n 和 m 个点的集合 A、B，不构造 $A(n)$和 $B(n)$，在 $O(n+m)$时间内可以找到这个交点，从而将 A 和 B 分割开。

第 8 章　算法的运动规划

S 计算几何中的一些研究问题来源于机器人学领域，这些问题称为运动规划或者更准确地称为算法的运动规划。本章研究该领域中的某些问题及其求解的算法。假设二维和三维空间中存在多个障碍物，这些障碍物呈多边形或者多面体。另外，一个可运动的物体从空间中的点 s 移动到另一点 t，运动的物体碰到障碍物时必须绕过它。这里可以提出许多问题，例如，求从 s 至 t 的最短路径；运动物体的形状是点、一条线段、一个凸多边形、一个圆或者任意多边形，求从 s 至 t 的路径等等。物体在运动过程中，要避免与所有障碍物的碰撞，也就是说，物体边缘上的任一点 a 与障碍物的一个内点重合时，就发生碰撞；点 a 沿障碍物边界的滑动是允许的。没有碰撞的路径称为自由路径。本章主要讨论以下三类问题：

(1) 判定问题：运动物体（即机器人）R 能够从 s 移动到 t 吗？即从 s 至 t 是否存在一条自由路径。

(2) 构造路径：寻找机器人 R 从 s 移动到 t 的一条自由路径。

(3) 最短路径：寻找机器人 R 从 s 移动到 t 的一条最短自由路径。

另外，当 R 具有不同几何外形时，可以再次提出上述三个问题。问题(1)比问题(2)容易（可以举出例子来说明这一点），而问题(2)又比问题(3)简单。问题(3)中“最短”的含义，一般是指自由路径长度最短。但进一步研究发现“最短”与 R 的外形也存在一定关系，如果 R 是一个圆，那么“最短”的含义是清楚的；而 R 是一条可以旋转的线段时，理解“最短”的含义就不那么简单了，这时先给出几个不同的定义，在此定义下求最短路径。

下面仅考虑几个最短路径问题：首先，机器人是一个点(8.1 节)。然后，研究两个容易理解的运动规划问题：平移一个凸多边形(8.3 节)；移动一个梯状物，或退化为一条线段(8.4 节)。另外，讨论移动铰链机器人臂(8.5 节)，最后考虑分离连接智力片的问题(8.6 节)。

8.1　最短路径

本节假设障碍物是多个离散的多边形，而且多边形顶点总数为 n，给定起点 s 和终点 t，s 和 t 不在任一障碍物多边形内部，问题是寻求 s 和 t 之间的最短路径。解决这个问题的一种方法，其思路是先求可视图，然后由可视图寻找最短路径。

8.1.1　可视图及其构造

一组多边形的可视图 $G=(V,E)$，其中结点 $v(\in V)$ 对应于多边形的顶点，边 $e(\in E)$ 对应于可以互相“看见”的顶点对 $(p_{ki}, p_{(k+1)i})$。注意，边 e 可能与多边形边重叠，这样，多边形的边也在可视图中。

从 s 至 t 的最短路径是由线段组成的一条折线，如图 8-1 所示，折线的两端点或者是 s、t，或者是多边形的顶点，也就是说，多边形的顶点可以取为起点和终点。这样，最短路径可以看成是障碍多边形顶点的序列，或者最短路径是由可视图（图 8-2）中边组成的序列。基于这个观察，有下面的结论。

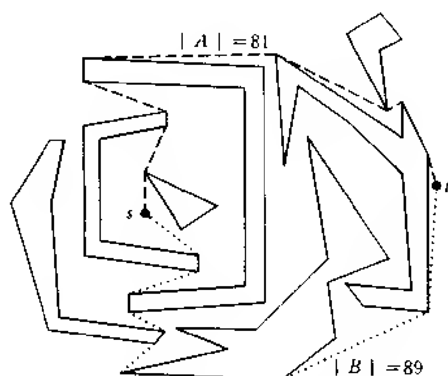


图 8-1 A 是从 s 至 t 的最短路径

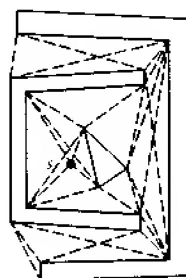


图 8-2 可视图

定理 8-1 最短路径是障碍多边形顶点可视图的一条子路径。

下面分三步证明该定理的正确性：

(1) 路径是折线的。假设相反，即路径不是折线的，该路径包含弯曲部分 C。这样，不可能是 C 的所有部分都沿着多边形边界，因为多边形边界不是弯曲的，因此，必定有一个不与任何多边形相接触并且可以用直线段代替的 C 的凸子部分，这与路径是最短的假设相矛盾。

(2) 路径的拐弯点是多边形的顶点：自由空间中的任何拐弯可以抄近路。

(3) 路径的线段是可视图的边：由可视的定义以及构成自由路径的定义可以推出。

由于可视图是有穷的，依据该定理，从 s 至 t 的最短路径必由一有穷路径集合中可以搜索得到。但该路径集合可以表示为多级多叉树结构，因此路径的数目可能以 n 的指数增长。为了获得有效的最短路径算法，先考虑可视图的构造。

构造一组多边形可视图的边几乎与寻找多边形对角线相同，唯一差别是：多个多边形与一个多边形，外部可视与内部可视。一种算法是对于不同多边形的每个顶点对 p_{ki} 和 $p_{(k+1)j}$，检查 $\overline{p_{ki}\ p_{(k+1)j}}$ 是否与多边形边相交，如果都不相交，则 $\overline{p_{ki}\ p_{(k+1)j}}$ 是可视图的一条边；否则，就不是可视图的边。由于顶点对数目可以达到 $O(n^2)$，而多边形边数为 $O(n)$，所以该算法的复杂性为 $O(n^3)$。另外，可视图边的数目为 $O(n^2)$，因此，$O(n^2)$ 是寻找可视图的任意算法的一个下界。利用第 7 章介绍的排列方法导致一个最佳算法，时间复杂性为 $O(n^2)$。经过长期的研究之后，Ghosh 和 Mount 找到了一个关于输出规模敏感的算法，复杂性是 $O(n\log n+|E|)$，当然，$|E|=O(n^2)$，但是多数情况下，$|E|$ 比 $\binom{n}{2}$ 小得多。

8.1.2 Dijkstra 算法

假设已构造出一组多边形的可视图，在该图中如何寻找出一条最短路径，这是图论中

所研究的一个问题：寻找加权图中的一条最短路径。加权是指边的长度，我们用欧几里德距离度量边的长度。Dijkstra 于 1959 年提出了解决这个问题的一个算法。介绍该算法之前先通过一个例子看其主要思想。

在图 8-2 中，设想可视边都是由红色液体在其内流动的细管组成，并且红色液体由 s 出发以相等的速率沿管道（可视图边）流向还是空的管道，随着时间的推移，被染红的管道越来越多，路径也越来越长，直到某时刻 g，终点 t 被着色。此时由 s 至 t 的一条最短路径被红色管道显示出来。

Dijkstra 算法的思想是模拟上述着色过程。设已知图 G 中最接近于始点 s 的 m 个结点，以及从始点 s 到这些结点中每一个结点的最短路（从 s 到其本身的最短路是零路，即没有弧的路，其长度为 0）。对始点 s 和这 m 个结点着色。然后，最接近于 s 的第 $m+1$ 个结点可如下求之：

对于每一个未着色的结点 y，考虑所有已着色的结点 x，把弧 (x,y) 接在从 s 到 x 的最短路后面，这样就得到从 s 到 y 的 m 条不同路。从这 m 条路中选出最短的路，它就是从 s 到 y 的最短路。相应的 y 点就是最接近于 s 的第 $m+1$ 个结点。因为所有弧的长度都是非负值，所以从 s 到最接近于 s 的第 $m+1$ 个结点的最短路必然只使用已着色的结点作为中间结点。

从 $m=0$ 开始，将这个过程重复进行下去，直至求得从 s 到 t 的最短路为止。

Dijkstra 最短路算法

步 1　开始所有弧和结点都未着色。对每个结点 x 指定一个数 $d(x)$，$d(x)$ 表示从 s 到 x 的最短路的长度（中间结点均已着色）。开始时，令 $d(s)=0$，$d(x)=\infty$（对所有 $x\neq s$）。y 表示已着色的最后一个结点。对始点 s 着色，令 $y=s$。$T\leftarrow\varnothing$。

步 2　对于每个未着色结点 x，重新定义 $d(x)$ 如下：

$$d(x) = \min\{d(x), d(y) + a(y,x)\}$$

其中 $a(y,x)$ 表示弧 (y,x) 的长度。对于所有未着色结点 x，如 $d(x)=\infty$，则算法终止。因为此时从 s 到任一未着色的结点都没有路。否则，对具有 $d(x)$ 最小值的未着色结点 x 进行着色。同时把弧 (y,x) 着色（指向结点 x 的弧只有一条被着色），即 $T\leftarrow$弧 (y,x)。令 $y=x$。

步 3　如果结点 t 已着色，则算法终止。这时已找到一条从 s 到 t 的最短路。如果 t 未着色，则转步 2。

注意，已着色的弧不能构成一个圈，而是构成一个根在 s 的树形图 T，此树形图称为最短路树形图。若 x 是最短路树形图中的任一结点，则从 s 到 x 的唯一的一条路是从 s 到 x 的最短路。

这个算法可以看成是根在始点 s 的树形图的生长过程。一旦到达终点 t，生长过程就停止。

例 8-1　给定有向图如图 8-3 所示，用 Dijkstra 算法找出从 s 到 t 的最短路径。

步 1　开始，只有 s 着色，$d(s)=0$。而且对于所有 $x\neq s$，$d(x)=\infty$。

步 2　$y=s$

$$d(a)=\min\{d(a),d(s)+a(s,a)\}$$

$=\min\{\infty,0+4\}=4$

$d(b)=\min\{d(b),d(s)+a(s,b)\}$

$=\min\{\infty,0+7\}=7$

$d(c)=\min\{d(c),d(s)+a(s,c)\}$

$=\min\{\infty,0+3\}=3$

$d(d)=\min\{d(d),d(s)+a(s,d)\}$

$=\min\{\infty,0+\infty\}=\infty$

$d(t)=\min\{d(t),d(s)+a(s,t)\}$

$=\min\{\infty,0+\infty\}=\infty$

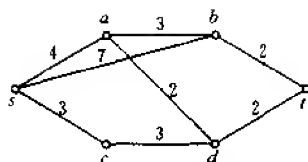


图 8-3 有向图

由于 $d(c)=3$ 是最小值，所以对 c 点着色，并对确定 $d(c)$ 的弧 (s,c) 着色，即 $T\leftarrow$ 弧 (s,c)。当前的最短路树形图 T 由弧 (s,c) 组成，如图 8-4(a)所示。结点 t 未着色，返回步 2。

步 2　$y=c$

$d(a)=\min\{d(a),d(c)+a(c,a)\}$

$=\min\{4,3+\infty\}=4$

$d(b)=\min\{d(b),d(c)+a(c,b)\}$

$=\min\{7,3+\infty\}=7$

$d(d)=\min\{d(d),d(c)+a(c,d)\}$

$=\min\{\infty,3+3\}=6$

$d(t)=\min\{d(t),d(c)+a(c,t)\}$

$=\min\{\infty,3+\infty\}=\infty$

由于 $d(a)=4$ 是最小值，所以对结点 a 着色，并对确定 $d(a)$ 的弧 (s,a) 着色，$T\leftarrow$ 弧 (s,a)。现有的最短路树形图 T 由弧 (s,c) 和 (s,a) 组成，如图 8-4(b)所示。

步 3　t 未着色，返回步 2。

步 2　$y=a$

$d(b)=\min\{d(b),d(a)+a(a,b)\}$

$=\min\{7,4+3\}=7$

$d(d)=\min\{d(d),d(a)+a(a,d)\}$

$=\min\{6,4+2\}=6$

$d(t)=\min\{d(t),d(a)+a(a,t)\}$

$=\min\{\infty,4+\infty\}=\infty$

$d(d)=6$ 是最小值，对 d 着色，确定 $d(d)$ 的弧有两条(即 (c,d) 和 (a,d))，可任选其中的一条，对其着色，我们选 (c,d)。这样，现有的最短路树形图 T 由弧 (s,c)，(s,a)，(c,d) 组成，如图 8-4(c)所示。

步 3　t 未着色，返回步 2。

步 2　$y=d$

$d(b)=\min\{d(b),d(d)+a(d,b)\}$

$=\min\{7,6+\infty\}=7$

$d(t)=\min\{d(t),d(d)+a(d,t)\}$

$=\min\{\infty,6+2\}=8$

$d(b)=7$ 是最小值，对点 b 着色，对确定 $d(b)$ 的弧 (s,b) 着色。现有最短路树形图 T 由弧 (s,a)，(s,c)，(c,d) 和 (s,b) 组成，如图 8-4(d)所示。

步 3　t 未着色，返回步 2。

步 2　$y=b$

$$d(t)=\min\{d(t),d(b)+a(b,t)\}$$
$$=\min\{8,7+2\}=8$$

对点 t 及弧 (d,t) 着色。最终的最短路树形图 T 由弧 (s,c)，(s,a)，(c,d)，(s,b) 和 (d,t) 组成，见图 8-4(e)所示。

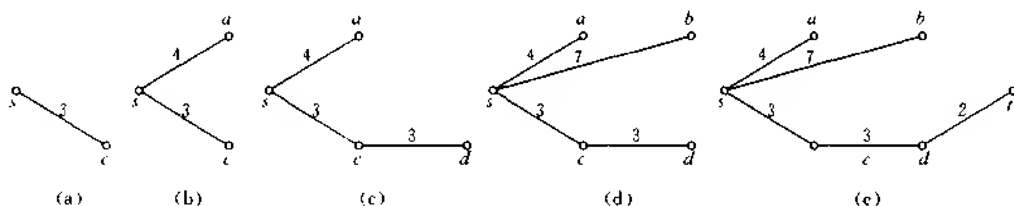


图 8-4　最短路树形图 T

从 s 到 t 的最短路由弧 (s,c)，(c,d) 和 (d,t) 组成，其长度为 $3+3+2=8$。

由于已假设多边形组有 n 个顶点，因此图 G 有 n 个结点，并且至多有 $\binom{n}{2}$ 条边。Dijkstra 算法每循环一次，处理 1 个结点，向 T 中加入 1 条边，所以 Dijkstra 算法至多需要循环 n 次。每次循环中要检验的候选边的数目大约是 $O(n^2)$，因为可视图可能有平方阶条边。这样，粗略分析 Dijkstra 算法的时间复杂性是 $O(n^3)$。如果注意到每次循环中不必重新检验这些边，那么在 $O(n^2)$ 时间内可以运行 Dijkstra 算法，加之可视图的构造需要 $O(n^2)$ 时间，故而有下面的定理。

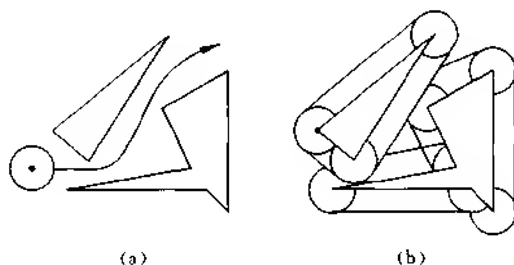
定理 8-2　在有 n 个顶点的多边形组中移动一个点，其最短路径可以在 $O(n^2)$ 时间和空间内找到。

8.2　移动圆盘

本节讨论运动规划的算法，目的是寻找任意路径(如果一条路径存在的话)，而不是最短路径。

假设机器人 R 是一个中心在 s 半径为 ρ 的圆盘，移动 R 一段距离之后，使 R 的中心位于 t，并且 R 不穿透任意障碍物。仍设障碍物是分离的多边形。图 8-5(a)中所示路径不是一条可通行的路径，因为机器人太宽，不能通过所指示的通道。下面的方法用于判定圆盘机器人 R 能否通过指定通道是非常有效的：设圆盘 R 的中心为 r，那么 r 不能离任意特定多边形 P 太近，也就是说，圆盘 R 在移动的过程中不可能移动到距 P 边的距离小于圆盘半径 ρ 的位置。这样，我们考虑一个扩展的障碍物 P'(P 向外扩充距离 ρ)以便移动点 r，如图 8-5(b)所示机器人 R 已收缩成一个点，并且把障碍物向外扩充 ρ，因此，上述问题简

化为 8.1 节中的问题。



(a) (b)

图 8-5 移动圆盘

在图 8-5(b)中,圆盘环绕 P 的边界移动时,圆盘中心 r 留下的轨迹便是 P' 的边界。如果 r 位于 P' 的外部,那么 R 与 P 不相交;否则就相交。显然,障碍物增大之后通道重叠,因此 R 无法通过图 8-5(a)中所指示的路径。以上描述 P' 的方式不清晰,下面介绍使用两个点集的闵科夫斯基(Minkowski)和的概念来描述 P' 将更明确。

给定平面上两个点集 S_1 和 S_2,如果在平面上建立一个坐标系,那么便可以把点看成该坐标系中的矢量。定义 S_1 与 S_2 的和:$S_1+S_2=\{x+y|x\in S_1,y\in S_2\}$,其中 $x+y$ 是点 x 和点 y 的矢量和,这称为 S_1 与 S_2 的闵科夫斯基和。另外,点 x 与集合 S_2 的闵科夫斯基和为 $x+S_2=\{x+y|y\in S_2\}$,因此,对于每个 $x\in S_1$,$S_1+S_2=\bigcup_{x\in S_1}(x+S_2)$是 S_2 的复制的并。现设 S_1 是一个多边形 P,并且 S_2 是中心在原点的圆盘 R,那么对于所有 $x\in P$,可以把 $P+R$ 看成是依据 x 转换的 R 的复制。由于 R 的中心在原点,$x+R$ 的中心将在 x,因此 $P+R$ 相当于放置中心在 P 的每个点之上的 R 的复制,故 $P+R$ 是 P 的扩展域 P'。

在图 8-5(b)中,考查三角形的扩展。当 x 是三角形的三个顶点时,在这些顶点处放置圆盘 R 的中心,这就完成了顶点处圆盘的复制工作。而当 x 位于三角形边界时,圆盘 R 的周边位于两端点圆盘的切线上,这就相当于圆盘中心沿三角形边界滑动时,R 的周边产生的轨迹。x 位于三角形内部时,$x+R$ 位于 P' 的内部。

给定一组离散的多边形和半径为 ρ 的一个圆盘,圆盘 R 的中心 r 放在始点 s 上,t 为终点,寻找由 s 至 t 的一条路径,使得圆盘 R 沿该路径从 s 可以移动到 t(即中心 r 落在 t 上)。下面叙述解决该问题的一种算法的粗略步骤。

寻找圆盘 R 从 s 移动到 t 的路径的算法

步 1 利用闵科夫斯基和及圆盘 R 扩大每个障碍物。

步 2 构造已扩大障碍物的并。

步 3 如果终点 t 与始点 s 位于平面上相同的部分之中,则由 s 至 t 有一条路径存在,并且通过改进可视图使之包含圆的弧可以找到最短路径;否则,s 与 t 之间不存在路径。

这个算法的时间复杂性为 $O(n^2\log n)$。

8.3 平移凸多边形

当机器人 R 是一个凸多边形时,机器人可以采取旋转的运动方式从一个位置移到另

一个位置，但本节将机器人的运动方式限制为平移。显然，凸多边形比圆盘复杂些，但利用闵科夫斯基和扩展障碍物的思想仍然有效。先介绍一个简单例子，由这个例子可以说明算法的基本思想。

设机器人 R 是一个正方形，以该正方形左下角 r 为参考点；多边形 P 为如图 8-6 所示的五边形。当 R 围绕 P 的边界 ∂P 移动时，r 描划出 P' 的边界，规定 r 不能穿透平面域上的一个已扩展的障碍物。这里的情况与 8.2 节不同，P 的扩展是通过 r 点的运动来实现的，比如，点 r 沿边 e_1 运动时，不必扩展 P，也就是说，e_1 是 P 和 P' 的共同边界；R 沿边 e_2 运动时，以 R 的水平宽度为 P' 的边界；沿 e_5 运动时，以 R 的垂直高度为 P' 的边界；沿 e_3、e_4 运动及在凹点的情况如图 8-6 中虚线所示。

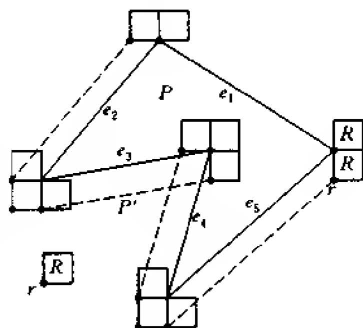


图 8-6　用 R 扩展 P 得到 P'

在 R 是一个圆盘的情况下，$P'=P+R$，其中“+”是闵科夫斯基和，但这个关系式在图 8-6 中显然不成立，例如，在 P 的边 e_1 处 $P+R$ 向外凸出，但 P' 不是这样。这里的计算是要通过参考点 r 用 R 的反射取 P 的闵科夫斯基和。因为对于闵科夫斯基和形式来说 r 是原点，R 的这种反射形式只不过是 $-R$，即求反 R 的每个点。因此图 8-6 中的 P' 是 $P+(-R)=P-R$。而圆盘关于其圆心是中心对称的，$R=-R$，因此这个新的形式与上一节的表示是一致的。因为闵科夫斯基减法就是反射域的加法，我们仍将称它为闵科夫斯基加法。

定理 8-3　设 R 是包含原点(参考点)的一个域(机器人)，而 P 是一个障碍物，那么在下述意义下 r 不可能穿透域 $P'=P-R$：

(1) 如果平移 R 使得 r 进入 P' 的内部，那么 R 穿透 P。

(2) 如果平移 R 使得 r 位于 $\partial P'$ 上，则 ∂R 与 ∂P 相切。

(3) 如果平移 R 使得 r 在 P' 的外部，则 $R\cap P=\varnothing$。

实际上，R 和 P 可以不是凸的，也不必是多边形。但本节仍设两者是多边形，并且 R 是凸的。

下面介绍构造两个多边形的闵科夫斯基和的方法，这里只是简要地叙述基本思想。我们仍然以图 8-6 所示 R 和 P 为例。图 8-7(b)示出 $P'=P-R$，并且 P 的边标记 1，2，3，4，5，$-R$ 的边标记 a，b，c，d，两者均按逆时针方向排列，用$\{1,2,3,4,5,a,b,c,d\}$中元素按照 P 或 $-R$ 的边产生 P' 的边的标记，因此，当 R 擦 P 的边 3 而过时，参考点 r 描画出 P' 的一条平行边，标记它为 3，而当 R 垂直地擦 P 的边 2 和边 3 交点而过时，标记 P' 的边为 c。

图 8-7(b)中已标记包含 P' 边的全部边界，包括 P' 内部凹顶点附近的边，这些边形成自身相交的多边形路径 τ。由 τ 可以找到 P' 的边界，有时称为 P 和 $-R$ 的圈。

可以把 P 和 $-R$ 的每条边看成一个向量，这些向量按逆时针方向排列，并且把它们都移动到一个公共点，如图 8-7(a)所示，称边向量的这种排列为“星”图。这种“星”图可以帮助我们理解 τ 边标记的序列。如果把 P 看成大的，R 看成小的，见图 8-7(b)所示，标记 τ 的序列为：1，b，2，c，d，3，a，b，4，c，d，5，a，其中 P 边(1，2，3，4，5)作为子序列，“星”图给出

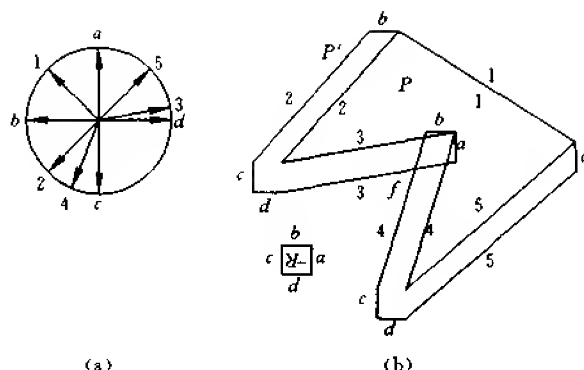


图 8-7 $P'=P-R$

(a) 边向量的星图 (b) 用 P 或者 $-R$ 标记作为 τ 的标记

$-R$ 的交替标记过程。现从 1 开始，逆时针方向环绕这个星旋转，在 P 边的标记 i 和 $i+1$ 之间记下所遇到的 $-R$ 的所有标记。例如，1 和 2 之间遇到 b，便产生子序列(1，b，2)；2 和 3 之间遇到 c 和 d，产生子序列(2，c，d，3)。继续下去，直至遇到 1 时就得到 τ 的整个序列。

剩下的问题是由 τ 序列构造 P' 的边界。一种方法是先求出多边形 P 的凹点，并在 τ 序列中找到与凹点关联的两条 P 边，然后求与该两条边相应边的交点，如图 8-7(b)中点 f，用点 f 代替 τ 序列中的 a，b 便得到 P' 的边界序列。假设多边形障碍物和机器人 R 都是凸的，那么计算 $P-R$ 的复杂性如下。

定理 8-4 如果 P 有 n 个顶点，而 R 的顶点数是一个常数，那么构造闵科夫斯基和 $P-R$的时间复杂性如下：

R	P	和的规模	时间复杂性
凸	凸	$O(n)$	$O(n)$
凸	非凸	$O(n)$	$O(n^2\log n)$
非凸	非凸	$O(n^2)$	$O(n^2\log n)$

下面给出凸多边形 R 规划运动的算法概要。设障碍物 $P_1,P_2,\cdots,P_m$ 共有 n 个顶点，算法描述如下：

步 1 增大所有的障碍物：$P_i'=P_i-R, i=\overline{1,m}$.

步 2 构造它们的并 $P'=\bigcup_i P_i', i=\overline{1,m}$.

步 3 寻找包含 s 和 t 的连通域，设为 k.

步 4 在 k 中寻找 s 和 t 之间的一条路径。

图 8-8 所示是一个例子，机器人 R 是一个四边形，其内部的任意一点选作参考点 r，图中有 7 个障碍物 $P_1,P_2,\cdots,P_7$。执行该算法之后，得到 3 个连通域 a，b 和 c。只要始点 s 和终点 t 位于同一连通域内，机器人 R 就可以由 s 移动到 t。因此，设计机器人的路径问题简化为在同一连通域内寻找参考点的一条路径。

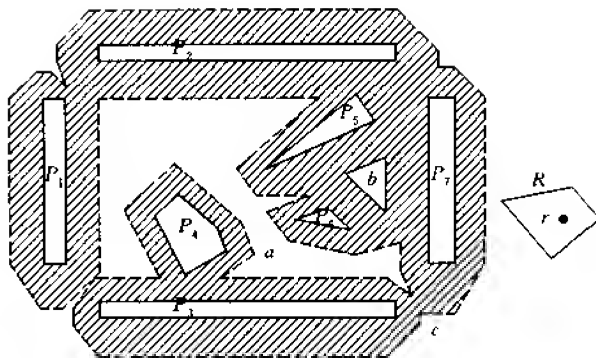


图 8-8　平移凸多边形 R 的例子

定理 8-5　设多边形障碍物顶点总数为 n，机器人 R 为凸多边形，寻找 R 由起点 s 到终点 t 的平移路径在 $O(n\log n)$ 时间内可以完成。如果 R 有 k 个顶点，则时间复杂性是 $O(kn\log(kn))$。

该定理由 Kedem 和 Sharir(1990)证明。

8.4　移动杆状机器人

在多边形障碍物中移动杆状机器人是运动规划问题中较复杂的一种。本节考虑的机器人在其运动过程中有三个自由度：水平平移、垂直平移和旋转，这意味着不可能把该问题的一个实例变换为二维空间中点的移动问题（如同 8.2 节和 8.3 节中所讨论的），因为这样一个点有两个自由度。然而，这个问题的实例变换成三维空间中移动点机器人的运动规划问题是可能的，将图 8-9 立体化之后便可以解释这一点。

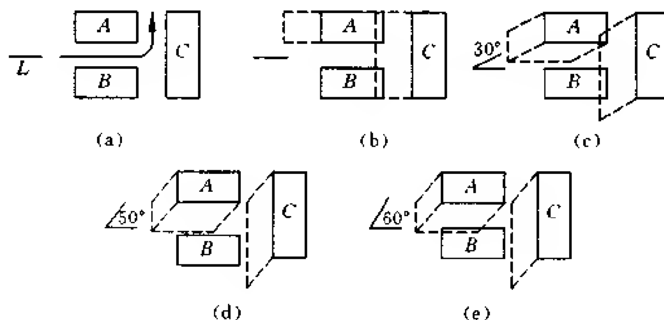


图 8-9　杆状机器人 L 旋转不同角度通过障碍物时的情况

图 8-9(a) 中，杆状机器人 L 先以水平状态，然后旋转 90°成垂直状态通过图中所示路径。图 8-9(b)示出对障碍物 A、C 通过水平线段 L 取闵科夫斯基和，得到扩展的障碍物，由图中易见它们是相互重叠的，这说明如果 L 不旋转，便无法通过通道。图 8-9(c)，(d)，(e)分别示出将 L 旋转角 θ，其值分别为 30°，50°，60 时扩展的障碍物。在图 8-9(c)中，A 与

C 之间的垂直通道关闭；图 8-9(d)中，A 与 B 及 A 与 C 之间的通道均已开行，L(以左端点为参考点)可以通过图 8-9(a)中所示路径；图 8-9(e)中，A 与 B 之间的水平通道关闭，而 A 与 C 之间的垂直通道已打开。

现将 θ 为不同值时扩展的障碍物重叠起来便构成三维空间中的立体图形，如图 8-10 所示。在这个空间中点(x,y,θ)表示 L 的参考点的一个位置，平行于 xy 平面并距 xy 平面距离为 θ 的平面表示 L 旋转 θ。这样就可以把二维空间中移动杆状机器人的问题转换为三维空间中移动点机器人问题，该空间称为杆状机器人的构形空间。

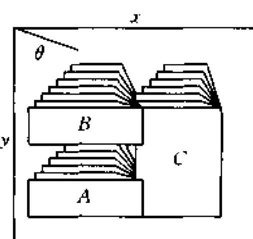


图 8-10　杆状机器人的构形空间

图 8-10 中，随着 θ 值的变化，产生的障碍物形状复杂，它不是多边形的(而在每个 θ 平面上它们是多边形的)，而是沿 θ 方向盘旋，就像盘旋的楼梯。参考点可以随意移动的空间称为自由空间。杆状机器人存在一条路径，当且仅当终点 t 与始点 s 同处于自由空间的同一连通域内。

利用这个方法可以得到多边形(而非矩形)障碍物中任意多边形机器人(不是杆状机器人)的构形空间。这就是说，上述基本思想可以推广到三维机器人和障碍物，甚至推广到有关节的机器人。

虽然建立构形空间的表达式以及寻找它内部的一条路径是一件复杂的工作，但由于这件工作的重要性，从而促使人们去研究它，并且的确构造出非常复杂的构形空间(有时高达六维空间)。在构形空间的基础上就可以寻找相应机器人的路径。Brost(1991)已构造出各种构形空间。

下面通过寻找构形空间中的一条路径，叙述求解运动规划问题的两种不同方法，并以杆状机器人为例进行说明。

8.4.1　网格分解

求解运动规划问题的网格分解方法是由 Schwartz 和 Sharir 首先提出的，并成功地解决了各种运动规划问题，下面以杆状机器人的路径问题为例进行叙述。

网格分解方法是划分构形空间成若干网格，并且通过寻找网格之间的一条路径确定构形空间里的一条路径。考虑图 8-11 所示环境：两个三角形和一个开的多边形，杆状机器人 L 是水平的，参考点在左端(箭头)。在自由空间的适当构形空间内网格是一个连通域。我们暂只考虑 L 呈水平方位，这样，构形空间只是平面的，而自由空间是用 L 通过闵科夫斯基和扩展障碍物之后剩下的空间。为了确定网格，给障碍物的边赋予标记 1，2，…，10，如图 8-11 所示。图中用∞表示包围无穷远的边。选 L 的箭头为参考点，水平地向左、向右移动 L，参考点碰到的障碍物边或平行边构成了网格。例如图 8-11 中，网格 A 有标记(3，2)，这表示 A 的左边是标记 3 的边，而右边是平行于边 2 并与边 2 距离为 L 的平行边；网格 B 有标记(3，8)；网格 C 有标记(1，9)；网格 D 有标记(1，8)。没有网格标记(3，6)，因为边 3 与边 6 之间的水平距离小于 L 的长度，使得 L 无法放入其中。图中斜纹表示的域是 L 的参考点可以随意移动的空间。

网格分解方法中，用图及连通图 $G(\theta)$表示网格结构，其中 θ 是 L 与水平方向的夹角，

即杆状机器人以参考点为旋转点转动的角度，$G(\theta)$表示 L 旋转 θ 角之后扩展的障碍物形成的网格。$G(\theta)$的结点是网格。如果两个网格的边界至少有一个公共点，则称该两个网格相碰撞。相碰撞的网格用一条弧连接相应的结点，这样，$G(\theta)$就是一个图。例如，图 8-11 所示的网格用 $G(0)$表示，代表网格 A 与 C 的结点之间无路径，而代表网格 A 与 D 的结点之间有路径。因此，图 $G(\theta)$中的路径可以转换成机器人 L 的路径。

如果将 L 绕参考点稍微旋转，图 8-11 中所示网格将改变形状，但网格的邻接性仍然保持，这就是说，连通图 $G(0)$不改变。而当旋转超过某个临界方位 θ^* 时，$G(\theta^*)$的网格结构将不同于 $G(0)$的网格结构。临界方位与障碍物边有密切关系。因为障碍物边 7 平行于 L，所以 $\theta=0$ 是临界方位，如图 8-11 所示。同理，图 8-12 所示也是临界方位，因为边 9 平行于 L。此时，网格 C 消失了，因为没有点有标记(1,9)，但产生了新的网格 E，标记(7,8)；网格 F，标记(5,8)；网格 H，标记(4,∞)。显而易见，临界方位必在 L 与某条障碍物边平行时才产生，而每条障碍物边由两个顶点确定，这样至多有 $O(n^2)$条边，因此也至多有 $O(n^2)$个临界方位。

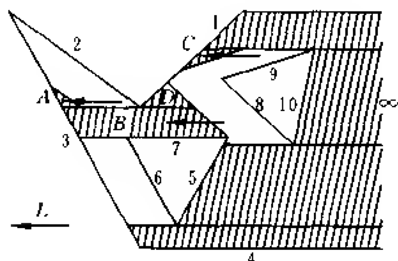


图 8-11 网格分解示例

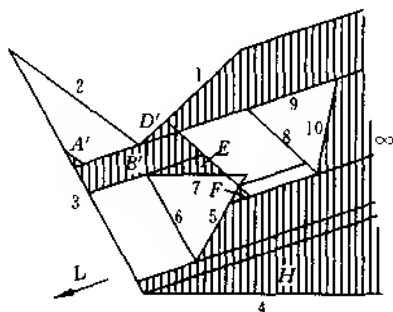


图 8-12 L 旋转之后的网格分解

现要构造连通图 G，它包含所有 $G(\theta)$图中的信息，为此推广网格定义以便表示三维构形空间的域，该域中的所有点具有与二维构形空间中同一点相同的标记偶对。这相当于把固定方位的网格按 θ 递增的顺序堆积起来，也就是由二维构形空间形成三维构形空间。这样，图 8-11 中网格 A 内的点与图 8-12 中网格 A'内的点都在相同的三维网格内。每个不同的三维网格是图 G 的一个结点，如果 G 中两个结点所代表的三维网格相碰撞，那么该两点之间用一条弧连接。

预置 $G \leftarrow G(0)$，然后按临界方位的分类顺序修改 $G(\theta)$，并把相关信息记入 G，这样便构造了图 G。构造图 G 是可能的，其细节请见 Leven 和 Sharir(1987)的论文。

此外，由 G 的一个结点所表示的单个网格内的运动规划问题，以及相碰撞网格之间的移动问题等都是容易求解的。例如，可以从一个网格的内部移动到它的边界，然后沿该边界移动到相邻网格的公共点。因此，运动规划问题简化为一个图问题：图 G 中寻找结点 s'(包含 s 的网格所对应的结点)与结点 t'(包含 t 的网格所对应的结点)之间的一条路径。如果 G 中不存在 s'与 t'之间的路径，那么就不存在杆状机器人 L 的路径；否则，由 s'与 t'之间的路径可以设计出 L 的运动路径。

8.4.2 收缩方法

求解运动规划问题的另一种方法是Ó'Dúnlaing 和 Yap(1985)提出的收缩方法。这里以杆状机器人 L 为例介绍这种方法的思想。

收缩方法的实质是构造关于 L 的 Voronoi 图，然后由 s 和 t 收缩到这个图，从而在该图的网格内完成路径规划。

首先解释关于 L 的 Voronoi 图是什么意思。对于 L 的一个固定的方位，定义与 L 有关的障碍物的 Voronoi 图是具有下述特征的自由点 x 的集合：选 x 为 L 的参考点时，L 至少与两个障碍物点等距离。由此要定义到 L 的距离。定义点 p 到 L 的距离 $d(p,L)$ 为

$$d(p,L) = \min_{x \in L} d(p,x)$$

其中 x 是 L 上任一点，$d(p,x)$ 是点 p 与点 x 之间的距离。与 L 距离为 r 的点 p 的轨迹是一个椭圆：两头是半圆，中间为两条平行线，如图 8-13 所示。图 8-13 中虚线表示 Voronoi 图，另外还有 L 的几个不同位置。例如，在位置 A，L 与边 3 和边 2 等距离；在位置 B，L 与边 1 和边 9 等距离；在位置 C，L 与边 5 和顶点 a 等距离，并且顶点 a 为边 8 和边 10 共有。图中从 A 到 B 无路径。

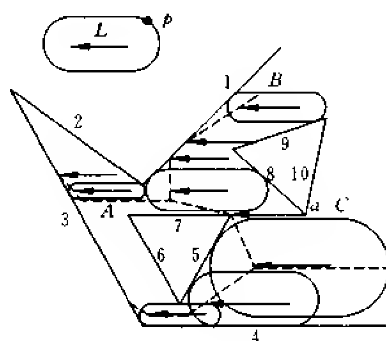


图 8-13 关于 L 的 Voronoi 图

当 L 的参考点沿图 8-13 中虚线(即 Voronoi 边)移动时，L 将处于距障碍物尽可能远的位置，所有这些位置与两个或者多个障碍物点等距离，这对于希望机器人避开与障碍物碰撞是非常有利的。

继而对于 θ 的每个值分别求出对应的 Voronoi 图，并将其重叠起来，从而构造出关于所有构形空间的 Voronoi 图。该图中可以见到由 Voronoi 边重叠形成的盘旋片(Voronoi 片)以及由 Voronoi 顶点重叠所形成的两片相交的棱。

仍将杆状机器人的运动规划问题转换成图问题。Voronoi 片之间的棱形成构形空间中 Voronoi 图上曲线的一个网格，这些曲线组成一个图 N：每条曲线是一条弧，曲线之间的交点是结点。

最后进行两次收缩：第 1 次收缩把 s 和 t 映射到 Voronoi 面(域)，而第 2 次收缩将 Voronoi 面(域)映射到网格。令 s' 和 t' 是网格上的这些收缩点，那么从 s 至 t 存在 L 的路径，当且仅当网格中从 s' 到 t' 有一条路径，该路径可以由搜索图 N 来确定。

已经证明存在求解二维空间中任意运动规划问题的算法，其复杂性为平方阶。设障碍物顶点数为 n，表 8-1 列出设计者及其算法的时间复杂性：

表 8-1

设计者		时间复杂性
Schwartz 和 Sharir	(1983a)	$O(n^5)$
O'Dunlaing, et. al	(1987)	$O(n^2\log n\ \log^* n)$
Leven 和 Sharir	(1987)	$O(n^2\log n)$
Sifrony 和 Sharir	(1987)	$O(n^2\log n)$
Vegter	(1990)	$O(n^2)$
O'Rourke	(1985b)	$\Omega(n^2)$

多面体障碍物中移动三维杆状机器人的问题将导致五维构形空间，这类问题相当复杂，现已有一些算法求解它，表 8-2 列出设计者及其算法的时间复杂性：

表 8-2

设计者		时间复杂性
Schwartz 和 Sharir	(1984)	$O(n^{11})$
Ke 和 O'Rourke	(1987)	$O(n^5\log n)$
Canny	(1987)	$O(n^5\log n)$
Ke 和 O'Rourke	(1988)	$\Omega(n^4)$

定理 8-6 运动自由度为 d 的机器人的任意运动规划问题可以在 $O(n^d\log n)$时间内求解。

该定理是最好的一般结果，但对于特殊问题也有较快的算法，例如，自由度为 3 的二维杆状机器人，按照定理 8-6 的结论应在 $O(n^3\log n)$时间内求解，但现已有 $O(n\log n)$的算法求解它。

8.5 机器人臂的运动

本节讨论平面多连杆臂运动，也就是平面机器人臂的运动，这是运动规划问题中的一个极其简单的例子。

将若干条线段 L_i 串联起来组成平面机器人臂，如图 8-14 所示，其中 J_i 是连接点(接头)，$i=0,1,\cdots,n$。J_0 叫做原点或者叫做臂的"肩"。J_n 是 L_n 的末梢，也称为手的尖端。J_i 是 L_i 和 L_{i+1}之间的接头。

表示机器人臂的另一种方法叫参数表示法。设 l_i 是连杆 L_i 的长度，j_i 是接头 J_i 处 L_i 与 L_{i+1}之间的夹角(按逆时针方向)，也就是向量$\overrightarrow{J_{i-1}J_i}$与$\overrightarrow{J_iJ_{i+1}}$之间的夹角，$j_0$ 是 x 轴正向

与$\overline{J_0J_1}$之间的夹角，j_n 未定义。机器人臂 A 由其连杆长度表$(l_1,l_2,\cdots,l_n)$确定。$j_i=0$ 时，称接头 J_i 为卡滞。

给定约束：(1)臂可自身相交(即不限制 j_i)；(2) 无障碍物。在这些约束下讨论可达性问题：已知组成臂 A 的 n 个连杆长分别为 $l_1,l_2,\cdots,l_n$ 以及平面上一点 p，要求确定 A 是否可以达到 p(称为可达性)。如果能够达到 p，则要求给出 $J_n=p$ 时的角序列 $j_0,j_1,\cdots,j_{n-1}$。后一个问题较前一个问题(判定问题)困难些。

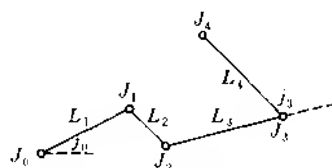


图 8-14　多连杆臂的表示法

1985 年，Hopcroft，Joseph 和 Whitesides 首先提出了机器人臂运动的算法问题，他们论证了无障碍物问题是易解的，而有障碍物问题是 NP 难的，但是限制机器人臂在圆内运动的问题是多项式的。后来人们改进了限制在圆内运动的算法或者得到不同障碍物环境下的类似算法。

8.5.1　可达性

多连杆臂可达到的点的集合是中心在原点(J_0)两个同心圆之间的环面。

设 $A_2=(l_1,l_2)$是由两个连杆组成的连杆臂。如果 $l_1\geqslant l_2$，则可达域是外部半径 $r_0=l_1+l_2$ 和内部半径 $r_i=l_1-l_2$ 的环面。如图 8-15(a)所示。如果 $l_1=l_2$，那么 $r_i=0$ 并且环面是半径为 r_0 的一个圆盘。

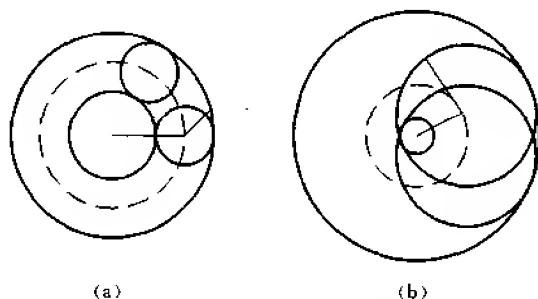

(a)　　　　(b)

图 8-15　2-连杆臂的可达域

(a) $l_1>l_2$　(b) $l_1<l_2$

当 $l_1<l_2$ 时，$r_0=l_1+l_2$ 但有 $r_i=|l_1-l_2|$，如图 8-15(b)所示。

可以把 2-连杆臂的可达域看成是两个圆的闵科夫斯基和：在半径为 l_1 的圆周 C_1 上的每个点处放置半径为 l_2 的圆的圆心。图 8-15(b)中虚线所示即半径为 l_2 的圆的圆心。这样，两个中心为原点的圆的和是一个中心定在原点的环面；此外，环面和圆(两者中心定在原点)的和是中心定在原点的环面。因此有下面的定理。

定理 8-7　一个 n-连杆臂的可达域是原点为中心的环面。

显然，环面的外部半径 $r_0=\sum_{i=1}^{n}l_i$，但内部半径如何计算就不明显了。如果 $l_M=\max(l_1,l_2,\cdots,l_n)$并且 $l_M>\sum_{\substack{j=1\\j\neq M}}^{n}l_j$，则 $r_i>0$。

定理 8-8 n-连杆臂的可达域与连杆的排列顺序无关。

证明 依据向量加法的可交换性可以证明。例如考虑图 8-16(a)所示 2-连杆臂的结构，平行四边形的另两条侧边达到同一个端点，而向量的排列顺序恰好相反。图 8-16(b)示出 3-连杆臂亦成立，对于 n-连杆臂结论成立。 证毕。

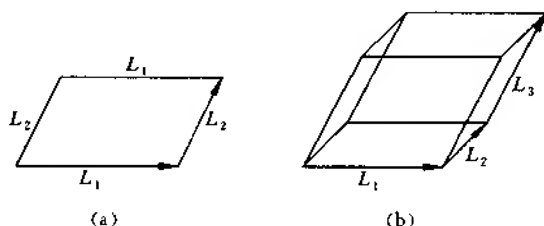


图 8-16 连杆顺序不影响可达域的说明
(a) 2-连杆臂 (b) 3-连杆臂

不失一般性，设 l_1 最大，那么 $r_i = l_1 - \sum_{i=2}^{n} l_i$。

定理 8-9 n-连杆臂的可达域是以原点为中心的环面，其外部半径 $r_o = \sum_{i=1}^{n} l_i$；如果 $l_M = \max(l_1, l_2, \cdots, l_n)$ 且 $l_M \leqslant \sum_{\substack{j=1 \\ j \neq M}}^{n} l_j$，则内部半径 $r_i = 0$，否则 $r_i = l_M - \sum_{\substack{j=1 \\ j \neq M}}^{n} l_j$。

由定理 8-9 推得，在 $O(n)$时间内能够确定可达域：计算 l_M 和 r_o 及 r_i，点 p 是可达的，当且仅当 $r_i \leqslant |p| \leqslant r_o$。下面介绍如何确定角序列 $j_0, j_1, \cdots, j_{n-1}$，使得 $J_n = p$。

8.5.2 构造可达性

对于 1-连杆臂来说，可达域是中心在原点半径为 l_1 的圆周 C，也就是说，只要 p 位于圆周 C 上，则 P 是可达的。2-连杆臂可达性问题也不难求解。令 p 是要达到的点，求中心在原点(J_0)半径为 l_1 的圆周 C_1，与中心在点 p 半径为 l_2 的圆周 C_2 的交，这时可能有 4 种情况：0 个、1 个、2 个或者无穷多个解，如图 8-17 所示。

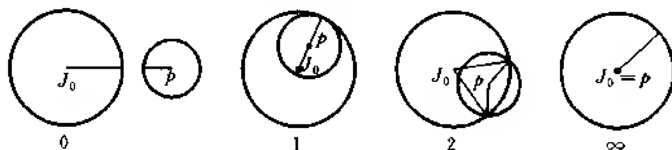


图 8-17 2-连杆臂可达性的四种情况

处理 3-连杆臂问题时，一般是将其转化为 2-连杆臂问题。令 $A_3 = (l_1, l_2, l_3)$。由定理 8-7 可知，$A_2 = (l_1, l_2)$的可达域是一个环面 R，R 的边界 ∂R 上的点在极端情况下满足 $l_1 + l_2$ 或者 $|l_1 - l_2|$。

为了把 3-连杆臂问题转化为 2-连杆臂问题，讨论 $p = J_3$ 为中心半径为 l_3 的圆周 C 与 ∂R 的交，有下列两种情况。

(1) $\partial R \cap C \neq \varnothing$，如图 8-18(a)，(b)所示。在这种情况下，通过调整图 8-18(a)或者逆调整图 8-18(b)的 L_1 和 L_2 可以把该问题转化为 2-连杆臂问题。我们使用调整 L_2 和 L_3 而不用逆调整 L_1 和 L_2。设 $\partial R = I \cup O$，其中 I 是环面的内部边界，O 是外部边界。如果

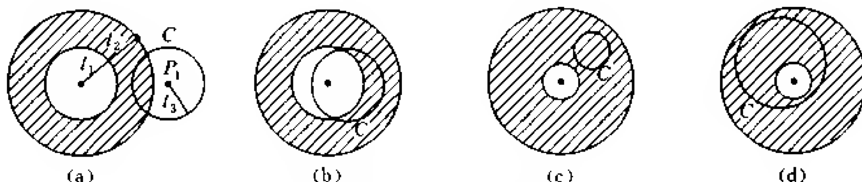


图 8-18　3-连杆臂可达性，阴影是环面 R，另一个圆是 C

$O \cap C = \varnothing$ 并且 $I \cap C \neq \varnothing$，如图 8-18(b)所示，选择正切于 C 半径为 l_2 的圆 C_2，通过 L_2 和 L_3 的调整而不是 L_1 和 L_2 的逆调整可以达到 p，如图 8-19 所示。

(2) $\partial R \cap C = \varnothing$，此时依据 C 是否包含原点 J_0 可以分为两种情况。

① C 不包含 J_0，如图 8-18(c)所示。令 C_2 是环面 R 中半径为 l_2 并且正切于 C 的圆，然后调整 L_2 和 L_3，便把问题转化为 2-连杆臂问题。

② C 包含 J_0，如图 8-18(d)所示。此时调整两个连杆的方法无效，现采用下述方法：任意选择 j_0（对 j_0 的每个值均有一个解）并作中心在 J_1 的圆 C_2。因为 C 在环面 R 中并且包含原点，所以它必定包围 I（R 的内部边界）。由于 C_2 把 R 的内部与外部边界连接起来，故它必在某处穿过 C，即 C_2 与 C 相交，对于已选择的 j_0 来说，由该交可以找到解。

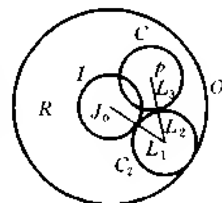


图 8-19　$C \cap I \neq \varnothing$ 时调整连杆 L_2 和 L_3

因此，总可以把这种情况转化为 2-连杆臂问题：任意选择 j_0，比如 $j_0 = 0$，然后求解所得到的 2-连杆臂问题。

把上述讨论概括成下面的定理。

定理 8-10　任一个 3-连杆臂问题都可以转化为下述 3 个 2-连杆臂问题之一：(1) $(l_1 + l_2, l_3)$；(2) $(l_1, l_2 + l_3)$；(3) $j_0 = 0$ 和 (l_2, l_3)。

证明　图 8-18(a)对应于(1)；图 8-18(b)和图 8-19 以及图 8-18(c)对应于(2)；图 8-18(d)对应于(3)。　　证毕。

可以把图 8-18 所示 3-连杆臂可达性推广到 n-连杆臂。设 R 表示 n-连杆臂 A 的 $n-1$ 个连杆的环面，另外 C 是中心在 p 半径为 l_n 的圆。分两种情况讨论。

(1) $\partial R \cap C \neq \varnothing$，如图 8-18(a)，(b)所示。选择两个交点之一作为 J_{n-1}。

(2) $R \supseteq C$，如图 8-18(c)、(d)所示。选择 C 上任意点 t，比如距 J_0 最远的点作为 J_{n-1}。

无论上述两种情况哪种发生，递归地寻找 $A_{n-1} = (l_1, \cdots, l_{n-1})$ 的构形以便达到 t。连接 t 与 p，并附连杆 L_n。递归的基础是 3-连杆臂问题的解。如果 n-连杆臂问题存在解，那么利用该递归过程可以找到一个解，其时间复杂性为 $O(n)$。这是由于常数时间内可以完成 n 减 1，包括计算 C 与 O 的交，C 与 I 的交，这里 $\partial R = I \cup O$。

对于任意 n-连杆臂问题，先利用定理 8-9 判定点 p 是否可以达到，如果能够达到，则利用上述递归过程寻找一个构形。

此外，有两种特殊情况：(1) $p\in O$；(2) $p\in I$。前者需要调整前 $n-1$ 个连杆。而后者仅在最长连杆两端处它们是弯折的，因此，在情况(1)下，臂不需要有多的弯折，而情况(2)下，p 可位于 C 上任何位置。

事实上，如果 n-连杆臂可以达到点 p，那么仅需弯折两次便能达到 p，并且容易确定具体的弯折位置。这表明任意 n-连杆臂问题可以转化为 3-连杆臂问题。

定理 8-11(两次弯折) 如果 n-连杆臂 A 可以达到点 p，那么它至多弯折两次便达到 p，即角序列 $j_0, j_1, \cdots, j_{n-1}$ 中有两个是非零角。选中间连杆 L_m 的两端为弯折位置，并且 $\sum_{i=1}^{m-1} l_i \leqslant \frac{l}{2}$，但是 $\sum_{i=1}^{m} l_i > \frac{l}{2}$，其中 l 是连杆总长度。

证明思想是通过卡滞所有接头 J_i(除了两个表示弯折的接头之外)，从而改进连杆臂 A，并证明所得到的新连杆臂 A' 具有相同的可达域。只要令 $j_i=0$，便可卡滞接头 J_i。由定理 8-9 知，r_0 仅依赖于连杆长度的和，这样的卡滞使 r_0 被确定，因此证明的目的是 r_i 未改变。下面分两种情况讨论。

(1) $r_i>0$，见图 8-20(a)。

由定理 8-9，仅当最长连杆 $L_M > \sum_{\substack{i=1 \\ i\neq M}}^{n} l_i$ 时，$r_i>0$，必有 $l_M > \frac{l}{2}$，因此 $L_M = L_m$，与它出现在连杆序列中的具体位置无关，因为无论连杆如何排列，连杆序列的中点都在 L_M 上。

如果卡滞所有接头(除 L_M 端点处的接头)，那么便构成一个新的连杆臂 A'，我们不改变 L_M 是最长连杆的事实(图 8-20(a)中，A 和 A' 的最长连杆的长度均为 6)。再次由定理 8-9，r_i 仅依赖于 l 和 l_M，所以 A' 与 A 有相同的可达域。

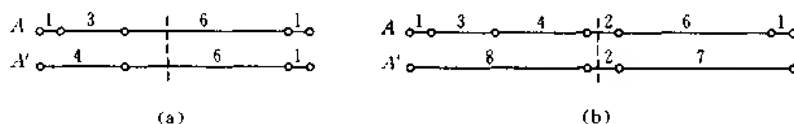


图 8-20 定理 8-11 的证明

(a) $l/2=5.5$，$r_i=1>0$，$L_M=L_m$　(b) $l/2=8.5$，$r_i=0$，L_m 不是最长的

(2) $r_i=0$，见图 8-20(b)。

在此情况下，由定理 8-9 知，$l_M \leqslant \frac{l}{2}$，因为 $l_M \leqslant \sum_{\substack{i=1 \\ i\neq M}}^{n} l_i$。令 L_m 是中间连杆并且卡滞 L_m 左、右侧的连杆，形成新的连杆臂 A'。在 A' 中可能改变最长连杆，如图 8-20(b)中，最长连杆的长度由 6 增至 8。但新的最长连杆 L_M' 的长度不可能超过 $l/2$，这是因为 L_m 包含长度的中点，它的左、右侧连杆序列的长度之和分别$\leqslant l/2$。因为仅当最长连杆的长度超过 $\frac{l}{2}$ 时，r_i 是非零的，但已设 r_i 为零，因此 A' 的可达域与 A 的可达域是相同的。　　证毕。

实际上，定理 8-11 给出一个构造可达性的 $O(n)$ 算法，其中依赖于 n 的唯一部分是计算 n 个连杆的长度，这之后该算法耗费常数时间。如果计数所执行的测试圆相交的次数，递归算法需要 $O(n)$ 时间，而两次弯折算法仅需 $O(1)$ 时间。因为在找到 L_m 之后，问题归结为一个 3-连杆臂问题，再用定理 8-10 可以将它转化为三个 2-连杆臂问题，每个执行一次

圆相交的测试。

为了实现上述算法，将连杆长度表存储在数组内，另外要解决如何判定圆相交的问题。圆的方程是一个2次方程，求两个圆的交点就是求两个2次方程组成的联立方程组的解，并且要判定出四种情况：0、1、2或无穷多个交点。具体实现步骤这里就不赘述了。

8.6 可分离性

可分离性在机器人学、线路设计与图形学中均受到很大的关注，这里要求物体彼此分离而不相互碰撞。给定平面上一组分离的多边形，每个多边形能够移动到无穷远而不与其他多边形碰撞吗？两个多边形碰撞的概念与前面的叙述相同。运动的类型限制于平移和旋转。另外，允许一个多边形移动还是多个多边形同时移动也应加以考虑。

8.6.1 多种可分离性

并不是所有多边形集合都是可分离的，即使不限制运动的类型，也难以保证多边形集合是可分离的。如图8-21(a)所示，它们是两个不可分离的互锁多边形（这里只限制于平面上的运动）。在允许旋转的情况下某些多边形集合是可分离的，但是如果仅限于平移，那就是不可分离的，如图8-21(b)所示。

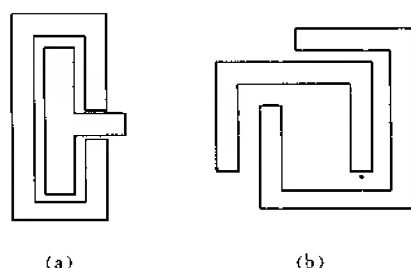


图 8-21

(a) 不可分离的多边形

(b) 用平移不可分离，但若使用旋转，则可以分离

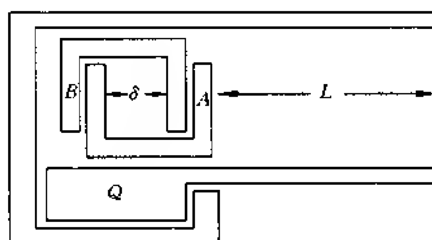


图8-22 多次交替移动 A 和 B

如果一次仅能移动一个多边形，而其他多边形固定不动，那么一组多边形可能是可分离的。可是也有经过多次移动才能分离的例子，比如图8-22所示，该图中要多次交替向右移动 A 和 B，然后 Q 向上再向右移动，便可分离多边形 A、B 和 Q。A 和 B 移动的次数 t 依赖于长度 L 和间隙 δ：$t \geq L/\delta$。显然，t 值可能相当大，并且不依赖于 n 和顶点数。这个例子没有说明该问题是困难的，但如果允许同时移动两个多边形(A 和 B)，那么分离一组多边形(A、B 和 Q)是十分容易的，而且移动次数也不多。

8.6.2 借助于平移的可分离性

1983年，Guibas 和 Yao 证明了一组凸多边形在下述条件下可以分离：(1)平移；(2)单向（所有平移朝相同的任意方向）平移；(3)每个多边形仅移动一次；(4)一次只移动

一个多边形。在这些限制条件下，图 8-23 所示多边形 A 或者 B（有一非凸多边形）沿方向 W 移动是不可分离的，这里只是对条件 2 略加限制。如果不是沿 W 方向移动，而是将 A 沿垂直向上的方向移动，则显然可以分离。另外，凸多边形在这些条件下沿任意方向是可分离的，但三维中的凸多面体在这些条件下不总是可分离的。

对于一组不相交的线段来说也可以提出相同的问题，即能否分离这组线段，也就是说，能否将线段集合中的线段向右沿水平方向移至无穷远处，由下面的定理可知答案是肯定的。

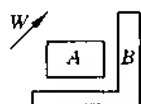


图 8-23 多边形沿 W 方向平移是不可分离的

定理 8-12 不相交的 n 条线段集合中，至少有一条线段从 $x=+\infty$ 处观察，它是可视的（可以看见整条线段）。

证明 考虑线段集合中线段 a 的上端点向右作的水平射线 L，如果 L 不与线段集合中其他线段相交，则称从 $x=+\infty$ 处观察，线段 a 的上端点是可视的。所有的这样的线段 a 组成子集 U，显然 U 不是空的；考虑其上端点的 y 坐标最大的线段，如果这样的线段只有一条，那么它就在 U 中；如果有多条，那么最右边的线段在 U 中。

U 中 y 坐标值最小的线段 b 必是可视的（从 $x=+\infty$ 处观察，可以看见整条线段 b），如若不然，则必有线段 c 遮住 b 的部分视线。设 c 的上端点的 y 坐标为 y_c，b 的上端点的 y 坐标为 y_b，则 $y_c \not< y_b$ 并且 $y_c \not> y_b$，因此不可能存在线段 c，即不可能有遮住线段 b 的线段 c，所以线段 b 是整条线段可视的。 证毕。

给定 n 个凸多边形集合 S，凸多边形 $P\in S$，能否水平移动 P 而不与 S 中其他凸多边形碰撞呢？这个问题可以转化为上述分离线段集合的问题：求出凸多边形 P 的最高顶点和最低顶点，设为 p_i 和 p_j，连接 p_i 与 p_j，用线段 $\overrightarrow{p_i p_j}$ 代替凸多边形 P。对 S 中其他凸多边形也作同样处理，得到线段集合 S'。如果 $\overrightarrow{p_i p_j}$ 沿水平方向平移时不会与 S' 中其他线段碰撞，那么凸多边形 P 也可以作同样的移动。因此，由定理 8-12 可以推得下面的结果。

定理 8-13 借助任意确定方向的平移可以分离平面上 n 个凸多边形集合，并且如果每次仅平移一个凸多边形，那么在 $O(n\log n)$ 时间内可以按序分离它们。

8.6.3 分离问题是 NP-难的

如果已知问题 A 是难解的，并且问题 A 可以归约到问题 B，那么问题 B 至少与问题 A 一样难。现在令问题 B 是分离问题：允许多边形平移并且每次只移动一个多边形，但每次可以朝不同方向平移，每个多边形可以移动多次。已知困难问题 A 是划分问题：给定整数集合 S，是否可以将 S 分成两部分 S_1 和 S_2，使得它们的和相等。比如，$S=\{1,3,3,5,6\}$，把 S 划分成 S_1 与 S_2，$S_1=\{1,3,5\}$，$S_2=\{3,6\}$，并且 $1+3+5=3+6$。但对于集合 $S=\{1,3,3,5,11\}$，答案是否定的。如果集合 S 有 n 个元素，那么划分 S 成两部分将有 2^n 种可能，显然时间耗费达到指数时间。已经证明划分问题是 NP-完全的。

给定划分问题的任何实例，构造一个可以求解的可分离问题，当且仅当划分问题可以求解。图 8-24 中给定集合 $S=\{1,3,3,5,6\}$ 的多个矩形表示，这些矩形的高度为 1，长度和恰好等于 S 中元素之和，即 18。多边形 Q 可以向下和向右平移，为了分离 D 和 Q，必须把矩形 1，3，3，5，6 移到左边的空闲空间去，而且折叠成长为 9 的两个长条放入空闲空间。因

此，多边形 D 和 Q 可以分离，当且仅当可以把矩形块压缩到左边空闲空间，当且仅当可以把 S 划分成两个相等的子集。这就证明了分离问题至少与划分问题一样困难，因此分离问题是 NP-难的。

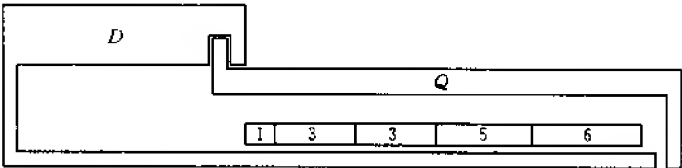


图 8-24　分离 D 和 Q，当且仅当划分问题可以求解

8.6.4　模拟河内塔问题

众所周知，河内塔问题的求解需要指数时间。现设计模拟河内塔问题的分离问题如下：如图 8-25 所示，用 U 形多边形代替金片，每个多边形的高为 h，厚度为 1，共计有 n 个 U 形多边形，这些 U 形多边形相互嵌套，相当于小金片放在大金片上面。另外，有三个空闲空间 A、B 和 C，相当于河内塔问题中的三根针。U 形多边形的移动规则如下：每次移动一个，并且限于平移，小 U 形嵌入大 U 形的内面。Q 可以向右和向下平移。为了分离 Q 与 G，必须将全部 U 形多边形从 A 移至 B 或者 C。显然，需要 2^n-1 次移动 U 形多边形才能完成将 n 个 U 形多边形从 A 移至 B 或者 C 的任务，然后可以极其容易地将 Q 与 G 分离。

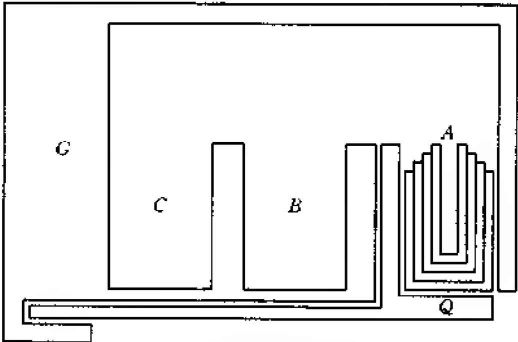


图 8-25　基于河内塔的分离难题

第9章　几何拓扑网络设计

前面介绍了凸壳、Voronoi 图、Delaunay 三角剖分及排列等 S 计算几何中的重要结构，另外还叙述了一些思想方法，如累接、修剪和搜索、几何变换、动态化、随机化、贪心法、分治法、平面扫描与轨迹等。本章将利用这些方法和几何结构解决几何拓扑网络设计(topological network design，即 TND)中的一些问题，也就是介绍解决 TND 中问题的几何算法。

一般说来，网络设计(ND)涉及通信、输送和分布式网络的设计、分析和综合，而 ND 问题由三类子问题组成：TND(涉及结点的数量和配置、结点之间互连边的表示以及两者成分的构形)，走线网络设计 RND(涉及并联网络各单元的顺序和走线)，适合网络设计 CND(涉及适应所要求流量的结点和边的最优规模)。

给定平面上 n 个点的点集 $S=\{p_1,p_2,\cdots,p_n\}$，点集 S 在平面有界域 E^2 内给定或者随机生成，并且已知各点的笛卡尔坐标(x_i,y_i)，$i=\overline{1,n}$。另设网络的总耗费是网络中流的线性函数，并且边的耗费与其长度成正比。还设 E^2 中障碍物集合 $\Omega=\{w_1,w_2,\cdots,w_k\}$，障碍物为凸多边形并且是分离的。我们规定距离函数是 L_p 测度的，其中坐标为(x_i,y_i)的点 p_i 与坐标为(x_j,y_j)的点 p_j 之间的距离为

$$d(p_i,p_j)=[|x_i-x_j|^p+|y_i-y_j|^p]^{1/p}$$

其中 p 是实数，$p\in[1,2]$。我们的任务是构造拓扑 $G(S,E,\Omega)$，用 $G(S,E,\Omega)$求解给定的问题，其中 S 是点集，E 为边集，Ω 是障碍物集。

计算几何中的思想方法和几何结构应用于某些 TND 问题已取得明显的成功，一个典型的例子是满足欧氏三角不等式的货郎担问题(ETSP)，用几何方法已找到平面上千百个点的 ETSP 的拟最优解，与传统的数学规划方法，组合优化方法相比较已显示出明显的优势。

本章介绍拓扑 $G(S)$结点定位和聚集问题；拓扑 $G(E)$边设计问题；$G(S,E)$结点和边设计问题；$G(\Omega)$问题。后者包括 $G(S,\Omega)$结点和障碍物设计问题，$G(E,\Omega)$边和障碍物设计问题，$G(S,E,\Omega)$结点、边和障碍物设计问题。

9.1　$G(S)$问题

$G(S)$问题是 TND 的重要组成部分，因为一旦定位新的装置，便开始建立道路、通信并运用与平面中其他 S 点相关的网络关系。因此，S 点的最优定位和聚集最优化是 TND 的核心。事实上，$G(S)$问题可以分为两个子问题类：

(1) 定位最优化问题　定位一个或几个新的点，使得与至 S 点距离有关的某些目标函数最小化(覆盖问题)或者最大化(间隙问题)。通常新点必须满足某些边约束，而涉及间隙问题的一类典型边约束是新点必须位于包含所有 S 点的某有界域内。

最简单的覆盖(间隙)问题是寻找一个新点 q,以 q 为圆心作圆,该圆是包含(不包含)所有 S 点的最小(最大)圆。

定位优化问题的目标函数通常是非线性的,人们习惯于用数学规划方法求解这些问题,但下面介绍的计算几何方法更是值得选择的解决该问题的一种途径。

(2) 聚集最优化问题　要求从几何上表示 S 点集的特征,使得能有效地查询最近的 S 点对或者到新点 q 的最近 S 点。关于特殊查询的点集 S 的几何特征通常是用划分空间成域的方法得到,因此,聚集最优化问题与可表示成这种划分的几何结构密切相关,而这种几何结构的构造是相对容易的。只要构造出这种几何结构,便能快速回答聚集最优化问题。

$G(S)$问题的分类见表 9-1。本节讨论两个有代表性的定位问题:最大间隙问题(MAX G)和最小覆盖问题(MIN C),另外还要讨论三个有代表性的聚集最优化问题:最近点对问题(CPP),所有最近邻近问题(ANNP)和邮局问题(POFP)。

表 9-1　$G(S)$问题分类表

<table>
<tr><th colspan="2">定位最优化问题</th><th colspan="2">聚集最优化问题</th></tr>
<tr><td>P 类</td><td>最大间隙问题(MAX G)
最小覆盖问题(MIN C)
最小轰炸问题(SBP)</td><td rowspan="3">P 类</td><td>最近点对问题(CPP)
所有最近邻近问题(ANNP)</td></tr>
<tr><td>NP 类</td><td>K-聚集问题(KCLP)
二次分配问题(QAP)</td><td rowspan="2">邮局问题(POFP)</td></tr>
<tr><td>NP?</td><td>多覆盖问题
多间隙问题
最小和韦伯问题(MSWP)</td></tr>
</table>

9.1.1　最大间隙问题(MAX G)

给定平面有界域 R 中点集 $S=\{p_1,p_2,\cdots,p_n\}$,非负权因子 $\{w_1,w_2,\cdots,w_n\}$ 并使用 L_p 测度,寻找点 $q(x,y)\in R$,使得 $\min\{w_i[|x_i-x|^p+|y_i-y|^p]^{1/p}\}$ 取最大值。这个问题又称为最大-最小问题。令所有 w_i 相等,利用点集 S 的 Voronoi 图可以求解这个问题:首先建立点集 S 的 Voronoi 图,然后以 Voronoi 点为圆心作圆,使圆内不含 S 中的点并且圆周上有 3 个 S 中的点(设平面上没有 4 点共圆),最后比较这些圆的半径,半径最长者对应的圆心即所求的点 q,如图 9-1 所示。

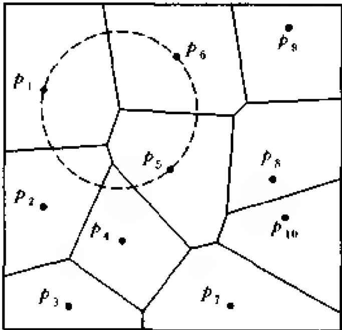


图 9-1　最大间隙问题的解

当所有加权因子相等时,最大-最小问题的最简形式(即最大间隙问题)产生了。当 $p=2$ 时,要求最大圆的圆心在 R 内并且圆的内部没有 S 点。不相等的加权因子表示现有的 S 点与新的不符合要求的点 q 之间的不相容性,这时的最大-最小问题比权相等时的 MAX G 问题困难得多。有界域 R 常常与凸壳 CH(S)相同。

所有 S 点位于 x 轴上时的 MAX G 问题在线性判定

树模型和代数判定树模型下需要 $\Omega(n\log n)$ 时间，但下面的算法只需要 $O(n)$ 时间：首先将最左和最右 S 点之间的区间划分成 $n-1$ 个相等长度的子区间 $L_i, i=\overline{1,n-1}$，把剩余的 $n-2$ 个 S 点置入适当的子区间内，根据鸽巢原理，必有一个子区间是空闲的（没有 S 点）。子区间 $L_i(i=\overline{1,n-1})$ 中最左和最右的 S 点分别存储在 LOW[i] 和 HIGH[i] 中。因此，最大间隙不可能是同一子区间内两个 S 点之间的部分。耗费 $O(n)$ 时间搜索 LOW 和 HIGH 可以求得最大间隙。遗憾的是，该算法不可能推广到二维。

当平面中 $R=\mathrm{CH}(S)$ 时，Shamos 设计了时间复杂性为 $O(n\log n)$ 的算法求解 MAX G 问题。该算法基于下述观察：新点 q 是 $\mathrm{Vor}(S)$ 的 Voronoi 点或者是 $\mathrm{Vor}(S)$ 与 $\mathrm{CH}(S)$ 的交点，这两种类型的点分别有 $O(n)$ 个。此外，如果 $\mathrm{Vor}(S)$ 是用双重连接表 DCEL 表示，那么在 $O(n)$ 时间内可以确定这些交点。

对于最大-最小问题有许多数学规划方法，Melachrinoudis 设计了求解一般权情况下的 $O(n^4)$ 算法，而对于 MAX G 问题，Dasarthy 和 White 提出了 $O(n^3)$ 的算法。

E^2 中的加权 Voronoi 图可以在 $\theta(n^2)$ 时间内构造。给定加权 Voronoi 图，在 Melachrinoudis 的 $O(n^4)$ 算法的基础上得到一个改进的算法。事实上，提供了下面的 $O(n^3)$ 算法，它基于构造 E^2 中加权 Voronoi 图的 $\theta(n^2)$ 算法。

局部最大值的候选点只可能出现于下列集合中：(1) R 中的 Voronoi 顶点集合；(2) Voronoi 边和 R 边的交点集合；(3) R 的顶点集合。考虑两个点 p_k 和 p_l，并且 $w_k>w_l$，则与 p_k, p_l 相关的 Voronoi 边（如果该边存在的话）不是圆的一条弧就是整个圆，其圆心在

$$o_{kl}=\frac{w_k^2p_k-w_l^2p_l}{w_k^2-w_l^2}$$

半径为

$$r_{kl}=\frac{w_kw_ld(p_k,p_l)}{w_k^2-w_l^2}$$

圆形弧的 Voronoi 边是由它的两个端点 e_k 和 e_l（即 Voronoi 顶点）来确定，并且它是由圆心 o_{kl} 来定向的。如果 $f_{kl}(x)=0$ 是过 e_k 和 e_l 的线的方程，那么圆形弧的 Voronoi 边 $\breve{e_ke_l}$ 是包含所有点 x 的弧，并且 $f_{kl}(o_{kl})f_{kl}(x)\leqslant 0$。

如果 Voronoi 顶点位于 R 中，那么它可能是局部最大值。另外，在 $O(\log m)$ 时间内可以判定顶点是否在 R 内（m 是凸壳的边的数目或顶点数目）。如果 R 中有 $O(n^2)$ 个 Voronoi 顶点，那么在 $O(n^2\log m)$ 时间内可以判定它们。

现考虑 Voronoi 边 $\breve{e_ke_l}$ 与 R 的第 j 条边的交点。如果 $f_j(e_k)f_j(e_l)<0$，则 $\breve{e_ke_l}$ 与线 $f_j(x)=0$ 有一个交点。如果该交点位于 R 的第 j 条边上，那么可以找到它。如果 $f_j(e_k)f_j(e_l)\geqslant 0$，则在 $f_j(x)=0$ 的情况下，$\breve{e_ke_l}$ 与 $f_j(x)=0$ 有 0、1 或者 2 个交点。

因为 R 的一条边与一条 Voronoi 边至多相交两次，并且有 $O(n^2)$ 条 Voronoi 边，所以与 R 的边有 $O(mn^2)$ 个交点。与 R 的顶点具有最短距离的装置点可以在 $O(n)+O(\log n)$ 时间内找到，对所有 R 的顶点耗费 $O(mn)+O(m\log n)$ 时间可以找到装置点。

总之，可以检测局部最大值的候选点，并且在 $O(n^2\log m+mn^2+mn+m\log n)=O(mn^2)$ 时间内计算全局最大值。

Melachrinoudis 和 Cullinane 提出了一个计算最大-最小问题的探索算法，其平均复杂性为 $O(n^2)$。该算法的步骤如下：

步 1 划分 R 为垂直窄条，将每个条内的 S 点按 y 坐标分类。

步 2 垂直条内，将每个 S 点与满足特定门限距离 r 的两个其他 S 点组合成三元组，对于每个可允许的三元组，可以找到由 Kuhn-Tucker 条件导出的圆锥方程的交，并且可以应用精确算法中有关最优性的测试。计算每个条中的局部最大值并存储在表中。

步 3 沿 R 的边界搜索局部最大值。

步 4 从步 2 和步 3 所找到的局部最大值中寻找全局最大值。

关于条的尺寸和数目以及 r 的确定请见 Melachrinoudis 和 Cullinane 的论文。该算法所求得的近似解很接近最优解，但可以进一步改进该算法，使之不仅可以得到更好的近似解，而且有较低的复杂性。

另外，周培德提出了求解该问题的一个算法，见 4.4 节。

9.1.2 最小覆盖问题(MIN C)

最小覆盖问题又称为最小-最大问题，它是最大-最小问题的对偶。当所有加权因子等于 1 时，出现最小覆盖问题 MIN C。而当 $p=2$ 时，该问题有明显的几何形式：给定平面点集 $S=\{p_1,p_2,\cdots,p_n\}$ 及非负权因子 $\{w_1,w_2,\cdots,w_n\}$，采用 L_p 测度确定 $p(x,y)$，使得

$$\max_{1\leq i\leq n}\{w_i[|x_i-x|^p+|y_i-y|^p]^{1/p}\}$$

是最小的，也就是在平面上寻求包含所有 S 点的最小圆，如图 9-2 所示。

$\Omega(n)$ 是 MIN C 问题的一个平凡下界。求解 MIN C 问题的一个简单方法是过 S 点的每个三元组(即三个 S 点)作一个圆，并验证剩余的 $n-3$ 个 S 点是否在圆内，这个算法的时间复杂性为 $O(n^4)$。Elzinga 和 Hearn 提出的算法将此限界改进到 $O(n^2)$，这个算法的主要思想是减少需要验证的三元组的数目。

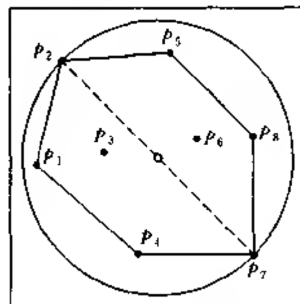


图 9-2 最小覆盖问题的解

包围所有 S 点的最小圆是唯一的。此外，该圆或者至少通过 3 个 S 点或者通过两个 S 点，并且这两个 S 点恰是圆直径的两个端点。Shamos 利用这个事实设计了一个复杂性为 $O(n\log n)$ 算法，下面描述该算法。

步 1 计算点集 S 的凸壳 CH(S)。

步 2 计算 CH(S)的直径，设为$\overline{p_ip_j}$。以$\overline{p_ip_j}$为直径作圆 C，如果 S 点都在圆 C 内，则圆 C 为所求最小圆；否则，转步 3。

步 3 计算点集 S 的最远点意义下的 Voronoi 图即 $\text{Vor}_{n-1}(S)$。

步 4 设 v 是 $\text{Vor}_{n-1}(S)$中的一个 Voronoi 点，以 v 为圆心，v 至 S 点集中 3 个最远点距离为半径作圆，该圆即所求。

步 1 在 $O(n\log n)$时间内可以完成，步 2 耗费 $O(n)$时间，步 3 为 $O(n\log n)$时间，步 4 的复杂性是 $O(n)$，因此该算法的时间复杂性为 $O(n\log n)$。

下面介绍周培德设计的一个算法。

$Z_{9\text{-}1}$算法(覆盖点集 S 的最小圆算法)

输入 域 $D=[0,A]^2$ 内的点集 $S=\{p_1,p_2,\cdots,p_n\}$。

输出 覆盖 S 的最小圆的圆心 o 与半径 r。

步 1 求点集 S 的凸壳顶点,设凸壳顶点 $C=\{p_1,p_2,\cdots,p_m\}$。

步 2 求 C 的直径,设为 l,l 的两个端点为 p_i 和 p_j。以 l 的中点为圆心 $o(k)$,$|l|/2$ 为半径 $r(k)$。

if $d(p_v,o(k))\leqslant r(k)$,$(v=\overline{1,m},v\neq i,j)$

then 输出 $o(k)$ 及 $r(k)$,终止。

else 转步 3。

步 3 计算凸壳顶点 $p_1,\cdots,p_{i-1},p_{i+1},\cdots,p_{j-1},p_{j+1},\cdots,p_m$ 与 l 的距离,用 $d(p_u,l)$ 表示顶点 p_u 与 l 的距离。

步 4 计算 $\max\limits_{1\leqslant u\leqslant m} d(p_u,l)$,最大值对应的 u 记为 k。

步 5 求 l 的中垂线与 $\overline{p_ip_k}$ 的中垂线,两条中垂线的交点 $o(k)$ 是圆心,$o(k)$ 到 p_i 的距离是半径,记为 $r(k)$。

步 6 **if** $d(p_v,o(k))\leqslant r(k)$,$(v=\overline{1,m},v\neq i,j,k)$

then 输出 $o(k)$ 及 $r(k)$,终止。

else if $d(p_{k'},o(k))>r(k)\wedge p_{k'}$ 与 p_k 在 l 同侧,$(k'=\overline{1,a})$

then 以 $p_{k'}$ 代替 p_k,重复第 5、6 步,直至不存在 $p_{k'}$。

else if $d(p_{k''},0(k))>r(k)\wedge p_{k''}$ 与 $p_{k'}$(或 p_k)在 l 异侧 $\wedge\min\limits_{k''}(\angle p_ip_{k''}p_{k'})$,$(k''=\overline{1,b})$,设以 $p_{k''}$ 为顶点的 $\angle p_ip_{k''}p_{k'}$ 最小。

then 过点 $p_{k'},p_{k''},p_i(|p_ip_{k'}|>|p_jp_{k'}|)$ 作圆,输出该圆,终止。

引理 9-1 设 $\overline{AB}$ 是圆 o 的一条弦,点 C 在圆周上而点 D 在圆外(见图 9-3),则 $\angle ACB>\angle ADB$。

证明 设 $\overline{AD}$ 与圆周交于 D',连接 D' 与 B,由平面几何知 $\angle AD'B=\angle ACB$,而 $\angle AD'B=\angle D'DB+\angle D'BD$。因此,$\angle AD'B>\angle ADB$,即 $\angle ACB>\angle ADB$。

推论 9-1 如果点 D 位于圆内(D 与 C 在 $\overline{AB}$ 同一侧),则有 $\angle ACB<\angle ADB$。反之亦成立。

引理 9-2 设 $\overline{AB}$ 是圆 o 的一条弦,点 C 在圆周上而点 D 在圆外,并且 C 至 $\overline{AB}$ 的距离大于 D 至 $\overline{AB}$ 的距离,$|\overline{AB}|>|\overline{AD}|$,则点 C 包含在 $\triangle ABD$ 的外接圆中(见图 9-4)。

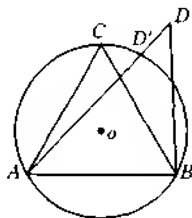


图 9-3 引理 9-1 的示意图

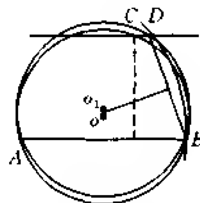


图 9-4 引理 9-2 的示意图

证明 由题设与引理 9-1 知,$\angle ADB<\angle ACB$,作 $\triangle ABD$ 的外接圆 o_1,根据推论 9-1,点 C

必在圆 o_1 内。

推论 9-2 圆 o_1 是包围点 A,B,C 与 D 的最小圆。

定理 9-1 设平面上点集 S 的 n 个点在域 D 中呈均匀分布，则 $Z_{9\text{-}1}$ 算法耗费 $O(m^2+n\log n)$ 次比较和 $O(n+m^2)$ 次乘法求出覆盖点集 S 的最小圆，其中 m 为点集 S 的凸壳的顶点数。

证明 $Z_{9\text{-}1}$ 算法首先求点集 S 的凸壳 C，S 中的点都位于 C 内，覆盖 C 的最小圆也覆盖 S 中的点。C 的直径 l，即 C 中距离最大的点对间的线段，以其中点为圆心，$|l|/2$ 为半径的圆，记为 o_1。如果该圆包围 S 中的所有点，则该圆是覆盖 S 的最小圆。因为直径是圆中最长的弦。

如果 o_1 没有包围 S 中的所有点，则先求出距 l 最远的点，设为 p_k，l 的两个端点记为 p_i 和 p_j，作$\triangle p_ip_jp_k$ 的外接圆，记为 o_2，如果圆 o_2 包围 S 中的所有点，则由简单的几何知识即知 o_2 是所求的最小圆。圆 o_2 没有包围 S 中的点的唯一可能情况是域 D_1（或 D_2）中有 S 中的点，比如 $p_{k'}$（或 $p_{k''}$）。域 D_1 是由过 p_k 平行于 l 的直线，以 p_i（或 p_j）为圆心，l 为半径的圆周及圆周 o_2 所围成的曲边三角形（域 D_2 类似）。步 6 的后半部分即为处理这种情况的，它稍微调整了圆心 $o(k)$ 及半径 $r(k)$，使 S 中的点全部落入圆内。由引理 9-1，引理 9-2 及推论 9-2 可推得，所求的圆是覆盖 S 的最小圆。

$Z_{9\text{-}1}$算法的步 1 最多需要 $O(n\log n)$ 次比较和 $O(n)$ 次乘法求得凸壳 C。步 2 耗费 $m(m-1)$次乘法与 $m(m-1)/2-1$ 次比较求出 C 的直径，再耗费 $2(m-2)$次乘法及$m-2$ 次比较确定 l 的中点是否可以为圆心。步 3 用 $7(m-2)$次乘法求凸壳各顶点至 l 的距离。步 4 耗费 $m-3$ 次比较确定 p_k。步 5 只要常数时间。步 6 的前半部分用 $2(m-3)$次乘法判定凸壳顶点是否在圆 o_2 内。由于 S 中的点呈均匀分布，所以落入曲边三角形 D_1（或 D_2）内的点极少，设为 a（或 b）个（$a<m,b<m$）。因此步 5，步 6 最多重复 a 次，即耗费$2a(m-3)$ 次乘法，另外还要求 b 次夹角并求出最小夹角即可得到最小圆。总之，$Z_{9\text{-}1}$算法需要乘法次数为

$$n + m(m - 1) + 2(m - 2) + 7(m - 2) + 2(m - 3) + 2a(m - 3) = O(n + m^2)$$

比较次数为

$$n\log n + m(m - 1)/2 - 1 + m - 2 + m - 3 = O(m^2 + n\log n)$$ 证毕

另外，通过把 MIN C 问题变换成 3 个变量二次规划问题的特殊形式之后，可以在 $\theta(n)$时间内求解 MIN C 问题。求解 2 个变量和 3 个变量的线性规划问题时可以使用修剪和搜索技术，见 5.3 节。

Megiddo 和 Dyer 已经证明，不必构造 n 条直线的凸交或者 n 个半平面的凸交也能求解 2 个变量或 3 个变量的线性规划问题。利用修剪和搜索技术可以删去所有多余的约束并在常数时间内找到最佳解。可以证明修剪过程的时间是 $O(n)$，因此 2 变量（3 变量）线性规划问题在 $\theta(n)$时间内可以求解。

引入新的变量 η，MIN C 问题变换成 3 个变量二次规划问题如下：

$$\text{minimize } \eta$$

约束条件为

$$\eta \geq (x_i - x)^2 + (y_i - y)^2, \forall\ p_i \in S$$

二次约束条件可以写成

$$\eta \geqslant -2x_i x - 2y_i y + (x_i^2 + y_i^2) + (x^2 + y^2), \forall p_i \in S$$

如果从二次约束条件中删去二次项(x^2+y^2)，便得到由线性约束定义的多面体，它的边界用$\eta=f(x,y)$来描述，其中$f(x,y)$是一个下凸函数，因为(x^2+y^2)是一个下凸函数，所以$f(x,y)+(x^2+y^2)$也是下凸函数。由$f(x,y)+(x^2+y^2)$所确定的面类似一个多面体的面。更准确地说，它的面是抛物面的面，并且它的顶点是抛物面三元组的交。为了定位最小值，只要删去多余的约束(抛物面)。3变量线性规划问题的$\theta(n)$修剪和搜索算法以直接方式推广到3变量二次规划问题的这种特殊情况。为了找到最小值，只需搜索与该顶点(由成对相交的抛物面形成的)相关的边。

最小轰炸问题(SBP)是与MIN C问题相关的问题，描述如下：给定平面点集$S=\{p_1, p_2, \cdots, p_n\}$，求至少包含$S$中$k$个点的最小圆。Shamos给出求解SBP问题的一个$O(n^4)$的算法。

9.1.3 最近对问题(CPP)

最近对问题是聚集最优化问题中的一个子问题。该问题描述如下：给定平面点集$S=\{p_1, p_2, \cdots, p_n\}$，求$S$中最近的点对。用分治法可以求解这个问题，如图9-5所示。

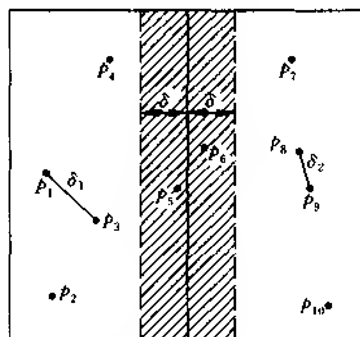


图9-5 分治法求解CPP问题

空中交通控制显示中确定最近的飞机对就是最近点对问题的一个应用。

为了确定CPP问题的下界，将判定n个实数集中是否包含两个相等数的问题变换为CPP问题，映射n个数到x轴上的点，并将这些点看成是S点(y坐标等于0)，然后求解CPP问题。如果最近点对之间的距离大于0(假设n个点的x坐标都不相等)，那么所有数都不相同。分类x轴上的n个数需要$O(n\log n)$时间，然后耗费$O(n)$时间可以求得最近点对。因此，$O(n\log n)$是求解CPP问题的一个下界。

平面上n个点至多可以组合成$\frac{n(n-1)}{2}=O(n^2)$点对，通过逐一比较所有的点对距离在$O(n^2)$时间内可以求解CPP问题。另外，下面介绍Bentley和Shamos设计的分治算法求解CPP问题，该算法的时间复杂性为$\theta(n\log n)$。

步1 用垂直线L划分S点集成两个近似相等的子集S_1和S_2，耗费时间$\theta(n)$。

步2 递归地求解S_1和S_2中最近点对问题，设$\delta=\min\{\delta_1, \delta_2\}$，其中$\delta_1$、$\delta_2$分别为$S_1$和$S_2$中最近点对之间的距离。

步3 按y坐标分类以L为中线宽度为2δ条状域内的点，分别记为S_1'和S_2'。如果全部S点按y坐标已分类，那么该工作耗费$O(n)$时间可以完成。

步4 搜索S_1'，对于S_1'中的每个点$p_i(x_i, y_i)$，在S_2'中检查y坐标在区间$(y_i-\delta, y_i+\delta)$内的点，由于S_1'、S_2'内的点已按y坐标分类，故寻找一个S点属于S_1而另一个S点属

于 S_2 的最近点对在 $O(n)$时间内可以完成。设 δ' 表示这种点对之间的距离。

步 5 选择具有距离 $\min\{\delta,\delta'\}$的点对作为解。

求解 CPP 问题的另一个 $O(n\log n)$算法是比较点集 S 的 Delaunay 三角剖分的边，其最短边即 CPP 问题的解。

9.1.4 所有最近邻近问题（ANNP）

所有最近邻近问题描述如下：给定平面上 n 个点的点集 $S=\{p_1,p_2,\cdots,p_n\}$，对每个 p_i 寻找与 p_i 最近的 S 点，即寻求点 p_j，使得 $d(p_i,p_j)=\min\limits_{p_k\in S-p_i}\{d(p_i,p_k)\}$，$i=\overline{1,n}$。如图 9-6 所示。

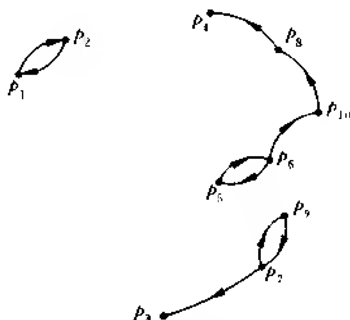


图 9-6 ANNP 的解

ANNP 问题是 CPP 问题的推广，因此，CPP 问题的下界 $\Omega(n\log n)$也可以作为 ANNP 问题的一个下界。

求解 ANNP 问题的一种算法是直接比较 $p_i(i=\overline{1,n})$至剩余 $n-1$ 个 S 点的距离，该算法的时间复杂性为 $O(n^2)$。另外，利用点集 S 的 Delaunay 三角剖分图可以求解 ANNP 问题，其复杂性为 $\theta(n\log n)$。对于 d 维空间($d\geqslant2$)中的 ANNP 问题，Bentley 和 Shamos 利用多维分治法设计了一个时间复杂性为 $O(n(\log n)^{d-1})$算法。

d 维空间($d\geqslant2$)中 ANNP 问题的求解方法的进一步改进是划分点集 S 所在域空间为小的立方体，使得每个小立方体内至多包含 S 中的一个点，求点 p_i 的最近邻近点只要在 p_i 所在立方体的邻接立方体中寻找。p_i 的第 k 个最近邻近点也可以用类似方法求解，时间复杂性为 $O(kn\log n)$。

9.1.5 邮局问题（POFP）

邮局问题描述如下：给定平面上 $S=\{p_1,p_2,\cdots,p_n\}$及询问点 q，求距 q 最近的 S 点。询问点 q 可能是一个或者多个。为了求解 POFP 问题，常常需要先将点集 S 进行预处理，比如将点集 S 三角剖分，或者构造 Vor(S)，以及在 Vor(S)基础上将平面划分成水平条、梯形等。如图 9-7 所示。

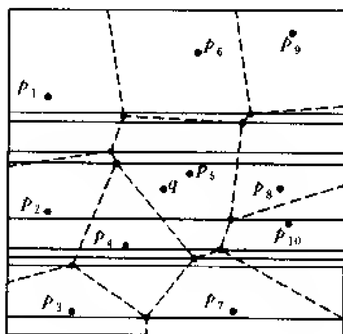


图 9-7 POFP 的解

POFP 问题已应用于信息检索（数据库查询）和模式识别（分类问题）。

如果询问点 q 的数目是有限的，通常不预处理 S 点，$\Omega(kn)$是一个平凡的下界，其中 k 为 q 点的个数。当 q 点的个数相当大时，为了有效地求得距 q 最近的 S 点，必须对 S 点集进行预处理。通常该预处理以某种几何结构形式产生，询问所需要的时间依赖于预处理阶段构造几何结构所需要的时间。

考虑一维空间中的 POFP(比如,S 点集全部位于 x 轴),不依赖于 S 点集的预处理过程的种类,寻找距点 q 最近的 S 点将需要 $\Omega(\log n)$时间。因此,$\Omega(\log n)$是任何维中询问时间的下界。

当维数 $d\geqslant 2$ 并且 q 点的个数相当大时,需要对 S 点集进行预处理,也就是把 S 点集所在的平面划分成域,使得落入相同子域的所有询问点有相同的最近 S 点。Voronoi 图是这类划分中的一种。在 $\theta(n\log n)$时间和 $\theta(n)$空间内可以构造 Voronoi 图。一旦构造出 Vor(S),再利用第一章中描述的任意点定位方法在 $\theta(\log n)$时间内可以确定包含询问点的子域。某些点定位方法可能要求进一步划分 Vor(S)成更小的子域。

当考虑其他距离标准和附加约束时,可以使用变形的 Voronoi 图。这时点定位工作通常也可以在 $\theta(\log n)$时间内实现。至于具体的点定位方法已在第 1 章中作了介绍,这里就不赘述了。

高维空间中 POFP 的求解更为困难。Dobkin 和 Lipton,Yao 等人提出的方法仍保持 $\theta(\log n)$询问时间,但要求 $O(n^{2^{d+1}})$空间并且另外要求预处理时间。对于 L_1 和 L_∞ 度量该限界仍然正确。$d=3$ 时,Chazelle 将空间和预处理时间减少到 $O(n^2)$并且询问时间为 $O((\log n)^2)$。另外,多维二叉搜索树可以用于求解 POFP。

9.2 $G(E)$问题

给定平面上 S 点集中点之间连线边 $E=\{e_1,e_2,\cdots,e_m\}$的集合,从 E 中选择边子集组成网络,使得在拓扑网络边的约束条件下目标函数极值化。一般情况下,设目标函数是网络边(被选取的边)的长度之和。通常要求构造路径(1 条或者多条不相交的路径)、树及回路等。

表 9-2 中列出 $G(E)$问题中的某些问题。表中的许多问题如何用几何方法求解还有待

表 9-2 G(E)问题分类表

树		路 径		回 路		生存网络	
P 类	欧几里德最小生成树(EMST) 直线最小生成树(RMST) 有向最小生成树(DMST) 欧几里德最大生成树(EMXST) ……	P 类	最短路径问题(SPP) ……	P 类		P 类	
NP 类	度数受约的 MST(δ-MST) 直径受约的 MST(ϕ-MST) 最短总路径长度 ST(SPST) 最优通信 ST(OCST) 第 k 个最好 ST(k-BST) ……	NP 类	漂移 TSP(WTSP) 最长路径问题(LPP) 权受约束的最短路径(WCSP) ……	NP 类	欧氏 TSP(ETSP) 瓶颈 TSP(BTSP) 中国邮路问题(CHPP) 乡村邮路问题(RUPP) 最长回路问题(LCP) ……	NP 类	多联网络 ……

进一步研究,树问题一般都有多项式时间算法求解。

本节叙述几何方法如何应用于 $G(E)$ 问题的求解,具体介绍求解 *EMST*、*ETSP*、*EMXST* 等问题的几何算法。

9.2.1 EMST 问题

欧几里德最小生成树 EMST 问题描述如下:给定平面上 n 个点的点集 $S=\{p_1,p_2,\cdots,p_n\}$,要求构造连接所有 S 点的生成树,并且长度最短。该生成树是一个网络,记为 $T(S)$。利用 Delaunay 三角剖分 DT(S)与 Voronoi 图 Vor(S)等几何结构可以求解 EMST 问题。如图 9-8 所示。

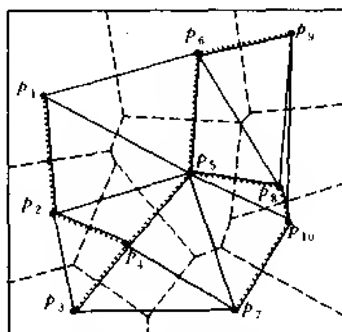


图 9-8 EMST 问题的解

EMST 问题已在通信、运输、网络、模式识别及 VLSI 等诸多领域中频繁出现,因此研究求解该问题的几何算法具有实际价值。

在线性时间内可以把分类问题变换到 EMST 问题,因此 $\Omega(n\log n)$是 EMST 问题的一个下界。事实上,该下界适用于任意维 d 中的所有 L_p-MST 问题,其中$d\geqslant 2$,$p\in[1,\infty]$。

下面介绍求解 EMST 问题的几个几何算法。

平面中 EMST 问题的一个 $\theta(n\log n)$算法基于下述观察:$T(S)$是 DT(S)的一个子图。因此,首先耗费 $O(n\log n)$时间构造 DT(S),然后用 $O(n)$时间可以找到 $T(S)$。

Chang 和 Lee 提出求解平面 EMST 问题的一种分治算法,该算法构造 DT(S)并在一次扫描中找到 $T(S)$的边。起初认为要求的最坏情况时间复杂性是 $\theta(n\log n)$,但后来证明该算法的复杂性是 $O(n^2\log n)$。因此,对于平面上的 EMST 问题是否存在不使用图算法而且复杂性为 $\theta(n\log n)$的算法仍是一个未解决的问题。

已经证明,RMST 和更一般的 L_p-MST,$p\in[1,\infty]$,可以通过构造直线 Voronoi 图的几何对偶来求解。直线 $T(S)$是 DT(S)图的子图。因此,在 $O(n)$时间内由 DT(S)可以求得 $T(S)$。构造直线 Voronoi 图在 $\theta(n\log n)$时间内可以完成。

当把平面中的 EMST 和 RMST 问题看成是已经求解的时候,仍然认为构造高维中的 $T(S)$是不完全清楚的。具体地说,不可能把基于构造 DT(S)的修剪方法和相关的邻近图推广到高维。

Bentley 和 Friedman 提出任意 $E^d(d\geqslant 2)$中的一个 EMSP 算法,后经变形,其最坏情况时间复杂性是 $O(n^2\log n)$。该算法粗略步骤如下:首先,解由 n 个子树组成,每棵子树是一孤立点,用一优先队列(堆)存储与某个 S 点 p_i 的距离按不减序排列的其他 S 点。其次,每次累接时,最小子树与它的最近的邻近连接,这时要合并两个优先队列。此外,合并优先队列时要修改某些入口。为了简化修改,保持 S 点的一个多维二叉搜索树。

Yao 提出 $E^d(d\geqslant 2)$中 EMST 问题的另一个算法,该算法修剪删去 $O(n)$条线段之外的全部线段,使得剩余的图至少包含一个 $T(S)$。这个算法也适用于 L_1-MST 和 L_∞-MST

问题。

Vaidya 提出一个修剪算法，将包含 S 点集的单位 d-立方体划分成边长 l_i 的相等小立方体。如果 i 级上的两个立方体 b_i^1 和 b_i^2 满足条件：(1)没有距离小于 $l_i/3$ 并且大于 l_i 的 S 点对(一个点位于 b_i^1 另一个位于 b_i^2)存在；(2)S 点集中不存在点 p，使得 p 分别至 b_i^1 和 b_i^2 中最远的 S 点的距离小于最近 S 点对(一个在 b_i^1，另一个在 b_i^2 中)之间的距离。那么 b_i^1 和 b_i^2 中的 S 点两两相邻。此外，属于 δ 级立方体 b_δ^1 和 b_δ^2 并且最近点对仅仅相距 l_δ 的 S 点成为相邻关系。修剪掉所有其他边。如果 δ 和 l_i 选择适当，那么可以证明剩余边集至少包含一个 $T(S)$。如果 S 点是随机均匀并且独立地分布于单位 d-立方体$[0,1]^d$ 中，那么可以证明修剪图是稀疏的概率是高的。该算法的期望复杂性是 $O(n\alpha(cn,n))$，其中 c 是依赖于 d 的一个常数，而 α 是逆 Ackermann 函数。

Agarwal 等人通过建立与双色最近点对(BCP)问题的关系提出一个 EMST 算法：在 E^d 中给定 n_r 个红点的集合和 n_b 个蓝点的集合，寻找一个红点 p_r 和一个蓝点 p_b，使得它们之间的距离是所有红-蓝点对之中最小的。EMST 算法划分边集成 $O(n\log^{d-1} n)$个子边集，每个子边集形成 S 的两个子集 S_r 和 S_b 上的一个完全双向子图。选择这样的子边集，使得 S_r 和 S_b 之间的唯一最近点对可能是任意 $T(S)$的一条边。为了求解给定 S_r 和 S_b 的 BCP 问题，可以使用一个点集的随机抽样划分初始问题成许多子问题。利用邮局问题的任意算法求解每个子问题。这个方法导致 E^3 中 BCP 问题的一个期望时间复杂性为 $O((n_r n_b \log n_r \log n_b)^{2/3}+n_r\log^2 n_b+n_b\log^2 n_r)$的算法。$d=3$ 时，复杂性为 $O((n\log n)^{4/3})$。

Vaidya 提出了一个与上述方法密切相关的探索方法，其基本思想是划分包含 S 点集的空间，在每次划分 P_i 中对于每个非空立方体 b_i，选择一个有代表性的 S 点 q_i。此外，划分是嵌套的并且较大立方体 b_i 中的点 q_i 也必须选作包含 q_i 的每个嵌套立方体的有代表性的点。如果在同一级 i 上两个有代表性的点至多相距 l_i，那么保留内连边。只要适当选择 δ 和 l_i，便可以证明保留下来的边必包含一棵生成树，该生成树的长度至多是 $T(S)$长度的$(1+\epsilon)$倍。此外，删去多余的边需要 $O(\epsilon^{-d}n\log n)$时间。该探索方法可以推广到具有任意 L_p-度量($p\in[1,\infty]$)的 d 维空间。

Clarkson 提出 $d=3$ 时的修剪-搜索探索法，求得的生成树的长度至多是 $T(S)$长度的$(1+\epsilon)$倍。它要求 $O(n(\log n+\frac{1}{\epsilon}\log\delta))$时间和 $O(n\log\delta)$空间，其中 δ=最远 S 点对之间的距离/最近 S 点对之间的距离。因此，该探索法的执行时间依赖于特殊 S 点的分布。

9.2.2 欧几里德 TSP

当使用离散的欧几里德度量时，已知 ETSP 在强意义下是 NP-完全的，但如果使用非离散的欧几里德度量，那么该问题仍然是 NP-难的。下面介绍求解 ETSP 的算法。

这里不介绍求解 ETSP 的各种指数算法，因为这些算法一般没有用到问题的几何结构。

多重分割探索法：开始时将所有 S 点集中的点作为部分解中的孤立点。然后，每次循环时，把结点数目小于 n 的非封闭回路并且结点度数不大于 2 的最短子路径加入到部分解中。在几何方案下，Bentley 证明，如果使用 MBST(多维二叉搜索树)，则该探索法的期

望时间复杂性是 $O(n\log n)$，没有最坏情况时间复杂性。但在图方案下，该探索法的最坏情况时间复杂性是 $O(n^2\log n)$。

最近插入探索法：开始时将单个 S 点作为一条初始旅行回路，每次循环时，把最接近于旅行回路的 S 点集中的 p 插入旅行回路，也就是用与 p 关联的两条边（该两条边的另一端点是已有旅行回路上一条边 e 的两个端点）代替原回路中的边 e。通常选择边 e 使得旅行回路长度的增量尽可能小。最远插入探索法是把离回路最远的 S 点集中的 p 插入回路。随机插入探索法，p 点是随机选取的。在图方案下，这些探索法的最坏情况时间复杂性是 $O(n^2)$，而在几何方案下，使用 MBST 将导致不超过 $O(n\log n)$ 的期望复杂性。

我们与其选择最接近于回路的 S 点集中的 p，不如寻求使回路长度增量最小化的点 p。这种改进的插入探索法所得到的回路长度至多两倍于最短回路长度。进一步改进探索法，选择点 p，使 $|\overline{p_ip}|+|\overline{p_jp}|-|\overline{p_ip_j}|$ 最小，其中 $\overline{p_ip_j}$ 是已有回路中的一条边，这种探索法求得的回路长度至多 $\lceil\log n\rceil+1$ 倍于最短回路长度。

最小生成树探索法是基于构造 EMST $T(S)$，沿 $T(S)$ 所有边的一侧移动之后再沿所有边的另一侧移动，这样便产生两倍于 $T(S)$ 边长之和的回路。此外，如果某个 p 点被访问的次数超过 1，那么用抄近路的方法减少多余访问的次数。可以证明，该探索法求得的回路长度至多两倍于最短回路的长度。在图方案下，该探索法的最坏情况下的时间复杂性是 $O(n^2)$，而在几何方案下，复杂性减少到 $O(n\log n)$。

匹配探索法：首先耗费 $\theta(n\log n)$ 时间确定 EMST $T(S)$。其次，复制 $T(S)$ 中的每一条边，把 $T(S)$ 变成一个欧拉图（在每两个奇度数顶点之间加一条边）。再次，对此欧拉图寻找一条欧拉回路。最后，利用抄近路的方法将欧拉回路变成 TSP 回路。可以证明这个要求 $O(n^{2.5}(\log n)^4)$ 时间的探索法（在图方案下要求 $O(n^3)$ 时间）产生的回路长度至多 $\frac{3}{2}$ 倍于最短回路的长度。

另一种探索法是从任意回路开始，不断地用其他的 $k(2\leqslant k\leqslant n)$ 条边代替回路中的 k 条边，使回路长度不断减少，直至回路长度不可能减少。

Karp 提出一种分治法，时间复杂性为 $O(n\log n)$。首先将所有 S 点集所在域划分成适当小的子域，使每个子域包含足够少的 S 点。其次在每个子域中用精确算法求部分回路。最后合并所有部分回路得到整体回路。当点独立、均匀分布于单位正方形时，这种方法得到的回路长度非常接近最短回路长度。

此外，Stewart 利用点集凸壳的边界作为初始子回路，得到的回路也较好。

9.2.3 欧几里德最大生成树问题（EMXT）

在图方案下，最大生成树问题与最小生成树问题密切相关：将边的权值（边长）取负号之后，就由一个问题转换成为另一个问题。而在几何方案下，由于明显的原因，无法应用删去 Delaunay 三角剖分边的方法。希望 EMXT 问题与最远点 Voronoi 图（及其对偶图）之间有一种关系，但遗憾的是两者都不是这种情况。

Monma 等人提出的算法要求 $O(n\log h)$ 时间和 $O(n)$ 空间，其中 h 是 S 点集凸壳顶点的数目。该算法首先寻求 S 点集中每个点 p 的最远配对。然后，用下述方法构造用 $F(S)$

表示的有向最远邻近网络：S 点集中每个点 p 作为网络的结点，而网络的有向弧连接每个 p 点与它的最远点配对。$F(S)$中的每个有向路径由弧的序列组成。对应于 $F(S)$中弧的所有边属于最大生成树。

令 $C_0, C_1, \cdots, C_{k-1}$表示 $F(S)$的弱成分。属于 $C_i(0 \leqslant i \leqslant k-1)$并且在 CH($S$)边界上的 p 点分成两个聚集 A_i 和 B_i，A_i 包含有向路径上编号为奇数的结点，而 B_i 包含有向路径上编号是偶数的结点。可以证明对于每个 $i(0 \leqslant i \leqslant k-1)$，$A_i \cap \mathrm{CH}(S)$和 $B_i \cap \mathrm{CH}(S)$是 CH(S)上的连续点。此外，A_i 和 $B_i(0 \leqslant i \leqslant k-1)$聚集的部分将按下面顺序出现在 CH($S$)上：$A_0 \cap \mathrm{CH}(S), A_1 \cap \mathrm{CH}(S), \cdots, A_{k-1} \cap \mathrm{CH}(S), B_0 \cap \mathrm{CH}(S), B_1 \cap \mathrm{CH}(S), \cdots, B_{k-1} \cap \mathrm{CH}(S)$。

不在 $F(S)$中的最大生成树的任意边至少必有一个端点在 CH(S)中，而且这样的边必须连接连续成分，因此只需确定每对连续成分之间的最长边(利用至少有一个端点是在 CH(S)中的事实可以缩小搜索范围)。这便产生连接 k 个成分 $C_0, C_1, \cdots, C_{k-1}$成一条回路的 k 条边。通过删去 k 条边中的最短边可以得到关于 S 的最大生成树。

该算法的时间复杂性由求CH(S)的时间确定。Kirkpatrick 和 Seidel 提出一种求平面点集 S 凸壳的算法，时间复杂性为 $\theta(n\log h)$，其中 h 为凸壳的顶点数目。因此，最大生成树算法的时间复杂性是 $\theta(n\log h)$。如果点集 S 中的所有点均在凸壳上，那么该算法的时间复杂性减少到 $O(n)$。这个算法推广到 d 维时，其时间复杂性是 $O((n\log n)^{2-\alpha(d)})$，其中 $\alpha(d)=2^{-(d+1)}$。$d=3$ 时，时间复杂性改进到 $O(n^{1.5}(\log n)^{2.5})$。

9.3 $G(S,E)$问题

给定平面点集 S，要求在关于拓扑网络边的约束条件下选择边的子集形成使某个目标函数极值化的网络。与 $G(E)$问题相反，这里允许增加某些被称为 Steiner 点的附加点，因而可能增加边的数目并且通常将导致更低耗费网络的产生。可以从预先给定的点集 S' 中选择 Steiner 点。换句话说，Steiner 点的数目和定位可能是未知的。典型的 $G(S,E)$问题是欧几里德和直线 Steiner 最小树问题。表 9-3 列出了 $G(S,E)$类中几何 TND 问题的分类。$G(E)$类中的所有问题可以形式化为 $G(S,E)$中的问题。此外，关于 Steiner 路径和 Steiner 回路问题的方法可以由它们的 $G(E)$对应物或多或少直接得到。反之，$G(E)$中的某些树问题是 P 类问题，而 Steiner 树问题似乎是固有的困难。

表 9-3　$G(S,E)$问题分类表

Steiner 树		Steiner 路径	Steiner 回路	Steiner 网络
NP 类	欧几里德 SMT(ESMT) 直线 SMT(RSMT) ……			

本节讨论求解几何 TND 问题的几何算法，这里要求增加某些新的点和边。ESMT (Euclidean Steiner Minimal Tree)和 RSMT (Rectilinear Steiner Minimal Tree)问题是

$G(S,E)$中的主要问题，下面介绍求解 ESMT，RSMT 问题的几何算法和探索算法。ESMT 和 RSMT 问题出现于许多实际问题中，因此，研究求解它们的算法是有实际意义的。

9.3.1 欧几里德 Steiner 最小树问题(ESMT)

ESMT 问题描述如下：给定平面上点集 $S=\{p_1,p_2,\cdots,p_n\}$，要求生成 S 的最短树 $T(S)$。与 EMST 问题不同之处是，增加的所谓 Steiner 点的集合 S' 可以位于平面上任何位置，只要它们能使总长度减少。利用 DT(S)和 Vor(S)可以求解这个问题，如图 9-9 所示，其中 $S'=\{p'_1,p'_2,p'_3\}$。

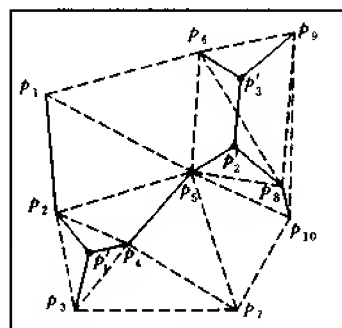


图 9-9 ESMT 问题的解

已知 ESMT 问题是 NP-难的，它的个别变形是 NP-完全的。下面介绍求解 ESMT 问题的算法。

平面中的 ESMT 问题的第一个算法是由 Melzak 提出的。在这个方法中，S 点集的每个子集都是分离的。

给定 S 点集中 p 个点的子集，$2\leqslant p\leqslant n$，存在完全的 Steiner 拓扑(其中所有 S 点的度数为 1，而所有 $p-2$ 个 Steiner 点 p' 的度数为 3)的一个有限数，该数通过检验 S 点的特殊定位可以简化为一个指数。

给定一个完全的 Steiner 拓扑，Hwang 提出了一个线性时间的几何算法，该算法或者产生(唯一的)具有这种拓扑并且在 Steiner 点 p' 处相交的 3 条边之间夹角均为 120° 的完全 Steiner 树，或者判定不存在这样的树。该算法由两个阶段组成，每个阶段包括 $p-2$ 次累接。在第一个阶段中，算法用(靠近 p' 的第 3 邻接的适当已定位的)等边点(充当一个 S 点集中的点)代替两个适当选择的 S 点集中的点以及它们共同的 Steiner 点 p'。该过程继续下去，直至得到具有两个点的完全 Steiner 拓扑。第二阶段中，算法反向执行并且一次确定一个 Steiner 点的位置。在此求解过程中，可以利用凸壳的几何性质。

给定所有完全的 Steiner 树，执行它们的一次无遗漏的连接，延伸所有 S 点的最短连接成为 $T(S)$。为了简化连接过程，Hwang 等人已提出各种不同的分解技术。

Hwang 和 Weng 提出可以用包含求解线性方程组的一阶段代数算法代替二阶段几何算法，该几何算法确定一个给定的完全 Steiner 拓扑的 Steiner 点定位。

关于 ESMT 问题的瓶颈算法似乎是完全 Steiner 树的连接。设 $D_S(T)$表示从完全 Steiner 拓扑 T 通过任何边序列收缩可以得到的所有拓扑(比如，删去边并抹去端点)，这里一个端点属于点集 S(把已抹去的点看作收缩后的一个 S 点)。Trietsch 和 Hwang 已发现，对于所有 n 个 S 点上至少一个完全 Steiner 拓扑 T 来说，关于 S 的 ESMT 有一拓扑(不必完全的)，它至少属于一个 $D_S(T)$。他们还提出了寻找属于拓扑 $D_S(T)$(如果存在的话)的所有树之中最短的负值边算法。遗憾的是，没有证明该算法在多项式时间内终止。最近 Hwang 和 Weng 提出了在 $O(n^2)$时间内求解该问题的一个算法，负值算法以及 Hwang 和 Weng 的算法是十分复杂的，这里将不描述。

E^d 中($d>2$)的 ESMT 问题比二维情况下更难求解，直到最近它也一直很少受到注

意。Gilbert 和 Pollak 已给出 E^d 中 ESMT 问题的几个基本性质(类似于 E^2 中的性质)。Smith 提出的一种枚举算法产生了 n 个 S 点上(以及 $n-2$ 个 Steiner 点)的所有完全 Steiner 拓扑。虽然 $d\geqslant3$ 时,没有精确算法确定给定完全 Steiner 拓扑的 Steiner 点的最优定位,但这样一棵树的长度是依赖于 Steiner 点定位的一个严格的凸函数。因此,可以应用数值方法。正如 Smith 所注意到的,设置偏导数为零,并且应用标准数值方法是无效的,因为长度函数在接近全局最小时处于非光滑的、复杂的状态。采用对其 3 个邻近的精确求解方法,重复修改每个 Steiner 点的定位将会更好些,但这种方法的收敛性可能非常慢。Smith 证明了利用第$(i+1)$次迭代中求解的下述 $2n-4$ 元线性方程组可能同时并适当地修改所有的 Steiner 点,

$$(x_k^{i+1},y_k^{i+1})=\left(\sum_{e_{kj}^i}\frac{x_j^{i+1}}{|e_{kj}^i|}\Big/\sum_{e_{kj}^i}\frac{1}{|e_{kj}^i|},\sum_{e_{kj}^i}\frac{y_j^{i+1}}{|e_{kj}^i|}\Big/\sum_{e_{kj}^i}\frac{1}{|e_{kj}^i|}\right),k=1,2,\cdots,n-2$$

其中每项和遍历邻接于第 k 个 Steiner 点的三条边,如果 Steiner 点的初始定位(x_k^0,y_k^0)构成 R^{2n-4}中的非 0 度量集,则解收敛于唯一的全局最小值。此外,解的序列是递减的,线性方程组可以在 $O(n)$时间内利用浮点运算和高斯消去法求解。试验结果表明收敛的速率在平均意义下是几何型的。

近来已证明 E^3 中等边四面体的无限聚集导致 E^3 中关于 Steiner 树比率的最好上界:

$$\rho_3=\inf_{S\in E^3}\rho(S),\text{其中 }\rho(S)=\frac{\mathrm{lSMT}(S)}{\mathrm{lMST}(S)}$$

计算 R-圆柱形得到 ρ_3,

$$\rho_3=\sqrt{\frac{283}{700}-\frac{3\sqrt{21}}{700}+\frac{9\sqrt{11-\sqrt{21}}\sqrt{2}}{140}}\approx0.7841\cdots$$

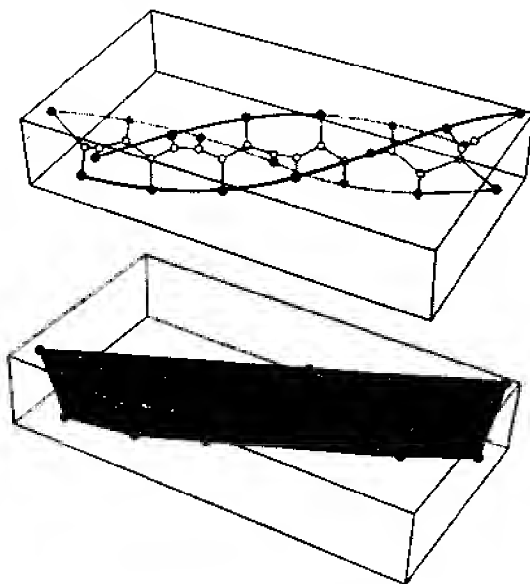


图 9-10 $N=20$ 螺旋线几何

我们将 R-圆柱形表示为三元螺旋线。图 9-10 表示 $N=20$ 个点的凸壳和 Steiner 树。该图明显地示出三元螺旋线结构。

估计 Steiner 点的任意结构的 ESMT 和 EMST 长度之间的比的下界是一个重要的问题。长期以来人们猜测没有问题的实例可以有少于 $\sqrt{3}/2=0.8666\cdots$的比例。后来，Du 和 Hwang 证明了一般情况下该猜测为真。

关于 ESMT 问题的几个探索方法是依据局部改进当前树 T 的思想，并通过已适当定位的 Steiner 点的引入和/或替换的方式。

Smith 提出了求解 ESMT 的 $O(n\log n)$的探索算法，该算法基于对 2、3 和 4 个 S 点适当选择连接的思想。利用 DT(S)与 Vor(S)可以识别 2、3 及 4 个 S 点集。更准确地说，在连接阶段只考虑规模 $k=2,3,4$ 的子集。

Beasley 设计了构造所有 3 个和 4 个 S 点集的 ESMT 的探索法，将这些树按照与它的 EMST 的比例进行分类并用贪心法连接得到生成所有 S 点的树。该探索法没有使用由 Vor(S)或 DT(S)提供的信息。但是后来 Beasley 又设计出建立在 DT(S)上的另一种探索法，并且利用模拟退火方法删去 MST(S)中冗余的 Steiner 点。该算法的最坏情况时间复杂性是 $O(n^{2.19})$，优于先前提到的所有探索法。

对于 E^3，一个基于 Delaunay 四面体剖分 DT(S)的 $O(n^2)$探索法已提出，该探索法的基本思想是建立在推测最优 R-圆柱形上，因为它为构造次最优解提供一个有效的分解结构。

当 S 点集中的点位于半径为 r 的圆周上时，S 的 ESMT 是由 n 边形周边删去其最长边所组成。当 S 点是正 n 边形顶点时，则对于 $n=3,4,5$，关于 S 的 ESMT 有完全拓扑。而对于 $n\geqslant 6$，S 的 ESMT 是由 n 边形的周边删去一条边形成的。

9.3.2 直线 Steiner 最小树问题(RSMT)

直线 Steiner 最小树问题描述如下：给定平面点集 $S=\{p_1,p_2,\cdots,p_n\}$，要求生成 S 的最短树 $T(S)$。与 EMST 问题相反，附加的 Steiner 点可以位于平面的任何位置，目标是使总长度减少。图 9-11 给出了一个例子。

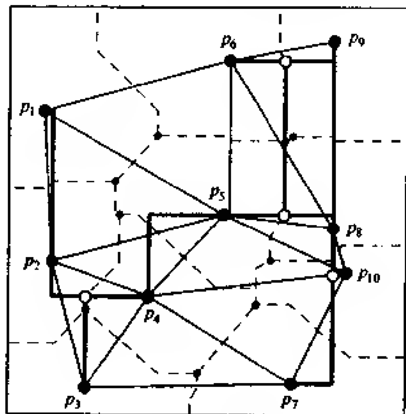


图 9-11 RSMT 的解

已知 RSMT 是 NP-难的，RSMT 的离散化变型是 NP-完全的。下面介绍求解 RSMT 问题的算法。

Hanan 证明 RSMT 问题至少有一个解由过 S 点的水平线、垂直线组成。因此，RSMT 问题的解通过图中 Steiner 问题的任意精确算法可以找到。Yang 和 Wing 设计出求解 RSMT 问题的分支限界算法，这是唯一的一个精确算法。

下面介绍求解 RSMT 问题的 5 种探索算法。

点集 S 的 RMST 是 RSMT 的一个简单的次最优解。Hwang 证明了对任意 S 点的结构，RSMT 和 RMST 长度之比不可能低于 2/3。

(1) Hwang 设计了复杂性为 $O(n\log n)$的探索法，该探索法基于 Lee 等人提出的 $O(n^2)$方法。这些探索法采用 3 个 S 点连接的方案。

(2) Smith 和 Liebman 提出一种局部改进的探索法，复杂性为 $O(n^4)$，他们还设计出基于 DT(S)、Vor(S)的 $O(n\log n)$探索法。

(3) Richards 提出 $O(n\log n)$的平面扫描探索法，S 点集中的点按 y 坐标不减序分类，然后从包含最小 y 坐标的 S 点的树 T 开始，后继 S 点按它们预先确定的顺序通过最短路径加入 T。

(4) Bern 和 Carvalho 研究了一种 $O(n^3)$探索法，该探索法由 n 个孤立点开始，然后将两棵最近的树互连起来。

(5) Komlos 和 Shing 提出一种分治探索法，它首先用垂直线与水平线将 S 点集所在的平面域划分成矩形并且用多维二叉搜索树表示，该树的每个叶结点至多含 t 个 S 点；然后确定每个矩形中的 RSMT；最后将这些解组合成一个总的次最优解。如果 $t=O(\log\log n)$，则该探索法要求 $O(n\log n)$时间。

如果 S 点集中的点均匀分布于矩形的边界上，Aho 等人设计出一个 $O(n^3)$的动态规划算法。当 S 点全在凸壳边界上时，Provan 和 Bern 分别独立地提出了 $O(n^6)$算法。

9.4 $G(\Omega)$问题

设障碍物集 $\Omega=\{w_1,w_2,\cdots,w_k\}$，其中 w_i 是多边形并且互不相交。上述三类问题可以推广为具有障碍物的相应问题，也就是说，$G(\Omega)$问题可以分成三个子类：

$G(S,\Omega)$问题：当定位与 S 点有关的新点时，障碍物通常是指被禁止的区域。此外，一般使用最短线的距离。在聚集问题中，当必须考虑障碍物时与以后的多重询问有关的空间细分变得更为复杂。

$G(E,\Omega)$问题：要求所构造的网络的边必须避开障碍物，因此这些边将不再是直线段而是多边形的链。

$G(S,E,\Omega)$问题：所构造的网络的边（多边形链）和附加的 Steiner 点必须避开障碍物。

上述问题在实际应用中具有重要的意义。$G(\Omega)$问题通常是非常困难的，因为目标函数常常是不连续的并且解空间是非凸的。

$G(S,\Omega)$，$G(E,\Omega)$和 $G(S,E,\Omega)$中的所有问题可以阐述为有障碍物的问题。表 9-4 列出了其中几个重要的问题。由于这些问题难以求解，故常常考虑用探索法来研究，目前已开始应用几何方法设计求解这些问题的多项式时间算法。

表 9-4　$G(\Omega)$问题的分类表

<table>
<tr><th>$G(S,\Omega)$</th><th colspan="2">$G(E,\Omega)$</th><th colspan="2">$G(S,E,\Omega)$(WP 类)</th></tr>
<tr><td rowspan="4">定位问题
最优化问题
聚集优化问题
……</td><td colspan="2">树</td><td rowspan="2">Steiner 树</td><td rowspan="2">ESMT
W/障碍 ESMTO
直线 RSMTO</td></tr>
<tr><td rowspan="2">路径(P 类)</td><td rowspan="2">最短路径
多边形内的 SPPO
E 最短路径 ESPO
直线 RSPO
……</td></tr>
<tr><td colspan="2">Steiner 路径
幸存 Steiner 网络</td></tr>
<tr><td colspan="2">回路
幸存网络</td><td colspan="2">Steiner 回路
……</td></tr>
</table>

9.4.1　有障碍物的最大空隙问题(MAX $G(\Omega)$)

有障碍物的最大空隙问题(MAX $G(\Omega)$)属于 $G(S,\Omega)$并可描述如下：给定平面中障碍物集 $\Omega=\{w_1,w_2,\cdots,w_n\}$，有界域 R 包含所有障碍物，要求点 $p(x,y)\in R$ 使得 p 至任意障碍物的最短距离最大，如图 9-12 所示。如果障碍物是半径为 r_i，中心在 $p_i(x_i,y_i)$的圆，$i=\overline{1,n}$，则 MAX $G(\Omega)$问题有下述代数形式：

最大化　ξ

约束条件　$\xi^2\leqslant(x-x_i)^2+(y-y_i)^2,\forall\ i$

$r_i^2\leqslant(x-x_i)^2+(y-y_i)^2,\forall\ i$

$(x,y)\in R$

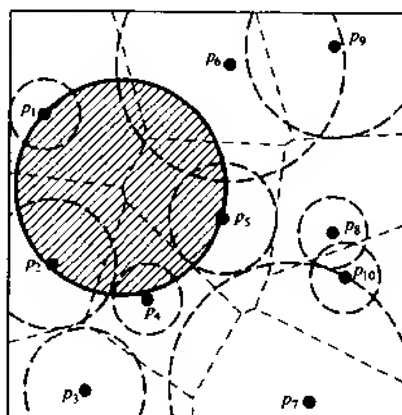


图 9-12　MAX G(Ω)的解

Melachrinoudis 和 Cullinane 提出了下述探索法：

步 1　不考虑圆约束情况下，利用松弛法求解无约束问题。

步 2　验证所有局部最大值是否满足未考虑的圆约束，如果最好的局部最大值满足圆约束，则终止。

步 3　寻找所有未满足的约束，并重复求解该问题，即利用 Kuhn-Tucker 条件的圆锥方程以及未考虑圆上的代数信息产生新的局部最大值并返回到步 2。

该方法不是单纯的几何方法，尽管如此，它还是基于格状分解并使用累接策略，首先忽略障碍物，然后逐步修正解。在关于障碍物问题的许多算法中这个方法是非常常见的。

9.4.2 具有障碍物的欧几里德最短路径问题(ESPO)

ESPO 可以描述如下：给定平面中两点 s 和 t 及多边形障碍物集 $\Omega=\{w_1, w_2, \cdots, w_k\}$，要求由 s 至 t 并避开所有障碍物的最短路径，如图 9-13 所示。

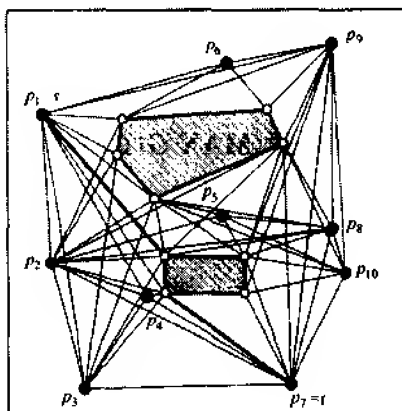


图 9-13 ESPO 的解

平面中的 ESPO 问题在多项式时间内可以求解，而 E^3 中具有多面体障碍物时确定最短路径长度的问题是 NP-难的。

由 s 到 t 的最短路径是一条折线链，该链的顶点是多边形障碍物的顶点。利用 Dijkstra 的最短路算法和可视图算法可以求解 ESPO 问题，其时间复杂性为 $O(n^2)$。如果可视图是稀疏的，则在 $O(m+n\log n)$时间内可以确定解，其中 m 是可视图边的数目。

利用最短路径映射 SPM(s,Ω)在 $O(n(k+\log n))$时间内求解任意多边形障碍物的 ESPO 问题的方法是由 Reif 和 Storer 提出的。如果给定 SPM(s,Ω)，则在 $O(\log n)$时间内可以确定包含 t 的域，而在 $O(b+\log n)$时间内能够确定到 t 的路径，其中 b 是路径上线段的数目。

Welzl 等人利用可视图给出了求解平面上 n 条线段的 ESPO 问题的算法，该算法要求 $O(n^2)$时间。不难修改这个算法使其能处理多边形障碍物，并且具有相同的时间复杂性。注意，如果使用可视图方法，那么对限界 $O(n^2)$将不可能改进。

多边形物体中两个物体(而非点)之间的最短路径的 $O(n^2)$算法是已知的。当 Ω 是平行线段集合时，Lee 和 Preparata 提出 $\theta(n\log n)$平面扫描算法。线段穿过扫描线并且把最短路径映射到扫描线。平面上没有最短路径的 $O(n^2)$算法能处理避开 n 条任意相交的线段。

Rohnert 给出平面中避开 k 个凸障碍物最短路径的 $O(n\log n+k^2)$时间的算法。这个时间限界在$O(k^2\log n+n)$时间和 $O(n+k^2)$空间预处理障碍物的条件下达到。预处理包括构造可视图的子图。Rohnert 还给出平面中避开 k 个凸障碍物最短路径的 $O(kn\log n)$时间

和 $O(n)$空间的算法。后者不需预先处理障碍物，而是利用 Dijkstra 最短路径算法在线计算可视性。当平面中有 k 个凸障碍物并且其边界至多相交两次时，Rohnert 给出的算法能找到平面中任意两点之间的最短路径，其时间复杂性为 $O(n\log n+k^2)$。这个时间限界在 $O(n\log n+k^3)$时间和 $O(n+k^2)$空间预处理障碍物的条件下达到。

对于 E^3 中有多面体障碍物的情况，Sharir 和 Schorr 提出一个 $O(n^n)$算法求解 ESPO 问题。Canny 将此限界改进到指数时间，他的算法允许障碍物不必是多面体，因此被移动的物体不必是点而可以是任意形状。

如果把问题加以限制，也就是说寻找 E^3 中点 s 与 t 之间避开单个凸多面体障碍物的最短路径，一个 $O(n^2\log n)$算法已由 Sharir 和 Schorr 提出。当凸多面体障碍物的数目增加到两个时，Baltsan 和 Sharir 提出了一个多项式时间算法（几乎与 $O(n^3\log n)$成比例）。Sharir 还证明了任意确定数目凸多面体障碍物中最短路径问题存在多项式时间算法。近来，Chiang 和 Tamassia 设计了一个简单多边形 P（相当于要回避的障碍物）内两个凸多边形之间最优的最短路径及最小连接路径查询的算法，令 n 是 P 的顶点数，h 为询问多边形顶点的总数。最短路径查询可以在时间 $O(\log h+\log n)$内完成，而最小连接路径查询在时间 $O(\log h+\log n)$内也可以最优地完成。

下面介绍探索方法。由于 E^3 中的 ESPO 问题的求解十分困难，故研究求解该问题的近似方法是有意义的。Papadimitriou 设计了一个多项式近似算法，求得的路径长度至多为最优路径长度的$(1+\varepsilon)$倍。Clarkson 提出另一个近似算法，该算法对于较大的 ε 将运行得更快些。

有若干问题与 ESPO 有关，比如，运动规划（第 8 章已介绍），简单多边形内部的最短路径，加权域问题，移动障碍物，有障碍物的直线最短路径问题（RSPO）等。

解决运动规划问题的多数方法与 ESPO 的可视图方法密切相关，另外还用到最短路径映射方法。

Lee 和 Preparata 设计出一个 $O(n\log\log n)$算法求解多边形内部的最短路径问题。先将多边形内部三角剖分，然后寻找包含 s 的三角形和包含 t 的三角形之间的三角形序列，该序列包含由 s 至 t 的最短路径。

将平面划分成多边形域，对每个域赋以适当的权，路径的耗费定义为权乘以路径通过不同域线段长度的和。所谓加权域问题就是要寻找 s 与 t 之间的最小耗费路径。ESPO 是该加权域问题的一种特殊情况：将权∞赋给障碍物域，而权 1 赋给平面的剩余部分，这个问题有许多实际应用。Mitchell 等人证明最小耗费路径由域内的直线段组成。他们还提出一个 $O(n^7L)$算法，其中 L 表示输入和输出位的数目。

Reif 和 Storer 设计允许障碍物移动的运动规划算法，这将更接近实际问题。

9.4.3 具有障碍物的 Steiner 最小树问题（ESMTO）

具有障碍物的 Steiner 最小树问题（Euclidean Steiner minimal tree problem with obstacles）可以形式化如下：给定平面上的点集 $S=\{p_1,p_2,\cdots,p_n\}$及多边形障碍物集 $\Omega=\{w_1,w_2,\cdots,w_k\}$，要求生成 S 的最短网络并且避开所有障碍物。利用 S 点集和障碍物顶点集的三角剖分可以帮助求解该问题。如图 9-14 所示，Smith 提出了一个求解该问题的探

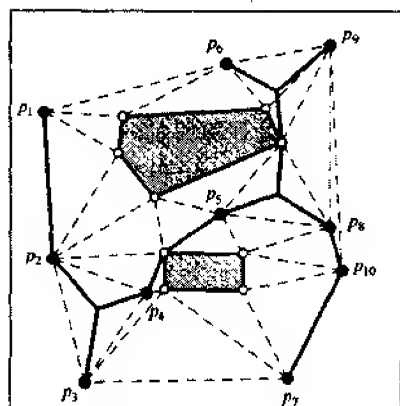


图 9-14 SMTO 的解

索法。

石油、天然气及光缆线路的设计中必然出现这个问题。

ESMTO 问题是 ESMT 问题的推广，故 SMTO 也是 NP-难的。因此没有多项式时间的精确算法求解 ESMTO。

Smith 给出单个凸多边形障碍物情况下三个 S 点的 $O(n\log n)$算法，其中 n 为 S 点和障碍物顶点的总数。后来，Winter 和 Smith 将此改进到 $\theta(n)$。如果在 $O(n)$时间内能预处理障碍物，则对任意 S 点的三元组 ESMTO 可以在 $O(\log n)$时间内求解。对于 4 个 S 点和一个凸多边形障碍物，Winter 和 Smith 提出一个 $O(n^2)$算法，在此情况下，目前还没有 $\theta(n)$算法。

Provan 设计了求解 ESMTO 问题的 ε-近似方法，步骤如下：

步 1 在 $O(n^3)$时间内构造路径凸壳 PCH(S,Ω)。删去 PCH(S,Ω)之外的障碍物，因为 ESMTO 的一个最优解必定在该壳的内部。

步 2 在 PCH(S,Ω)的内部（障碍物的外部）添加网格点（其密度依赖于 ε）的集合 S'，构造可视图 VG$(S\cup S',\Omega)$。

步 3 利用图中关于 Steiner 最小树问题的任意算法或探索法寻找 VG$(S\cup S',\Omega)$中 S 的 Steiner 最小树。

关于 ESMTO 问题，Armillotta 和 Mummolo 提出一个 $O(n^3)$的探索法，步骤如下：

步 1 构造可视图 VG(S,Ω)。

步 2 在生成所有 S 点的 VG(S,Ω)中找一个低耗费树 T。

步 3 改进 T：用三个末端点（不包括障碍物顶点）的欧几里德 Steiner 最小树代替关联边对。

步 4 如果 Steiner 点的某些关联边穿过障碍物，则修改 Steiner 点。如果无修改要求，则停止。否则转步 3。

Winter 和 Smith 提出与 ESMT 问题的 $O(n\log n)$探索法密切相关的另一种探索法，该探索法包括有约束的 S 点三角剖分以及求解具有一个或几乎没有凸多边形障碍物的 3

个和 4 个 S 点的 ESMTO。

推广 ESMTO 问题可以产生下述问题：

具有障碍物的直线 Steiner 最小树问题。Liestman 讨论了这个问题的求解方法，Watanabe 和 Sugiyama 设计出一个并行探索算法。

具有曲线障碍物的欧几里德 Steiner 最小生成树问题。修改 ESMTO 问题的算法可以求解这个问题。

黎曼曲面上的欧几里德 Steiner 最小树问题。Dolan 等人提出求解球面上 ESMT 问题的 $O(n\log n)$ 探索法。

有障碍物的三维欧几里德和直线 Steiner 最小树问题。这个问题有许多重要应用，比如 VLSI 设计中，元件和多层板之间的互连必须是三维的。

下面讨论几何结构（CH(S)，Vor(S)，DT(S)，VG(S)，CD(S)）、几何方法（累接、平面扫描、分治法、轨迹、修剪与搜索）与拓扑网络设计问题（$G(S,E,\Omega)$，$G(E,\Omega)$，$G(\Omega)$，$G(S,E)$，$G(E)$，$G(S)$）之间的关系。我们用笛卡尔坐标系的三条坐标轴分别表示几何方法、TND 问题与几何结构，如图 9-15 所示。该笛卡尔空间中的点（小立方体）表示该点对应的 TND 问题的求解使用了与该点相应的几何结构与几何方法。某些空白点表示至今还没有 TND 问题用到相应的几何结构和几何方法，有待人们去进一步研究，如表 9-5 所示（垂直于 y 轴切割空间），表中的空白格表示没有 $G(S,E,\Omega)$ 问题用到相应的几何结构和几何方法。

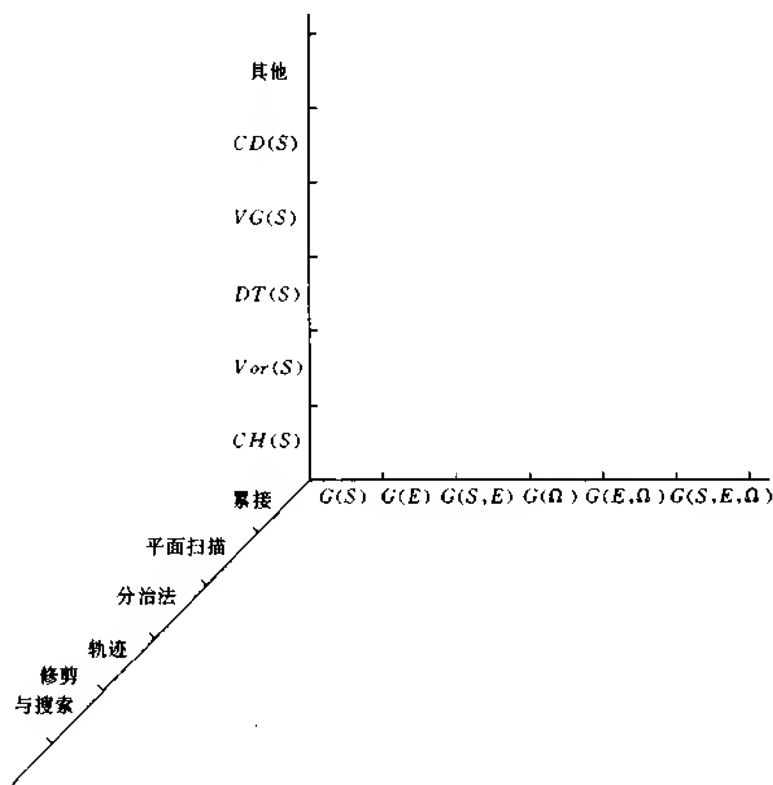


图 9-15　几何结构、几何方法与 TND 问题之间关系的表示

表 9-5 $G(S,E,\Omega)$分类表

几何方法 几何结构	累接	平面扫描	分治法	轨迹	修剪与搜索
CH(S)	ETSP:$O(?)$ RSMT:$O(n^4)$ RSMT:$O(n^6)$ ESMTO:$O(?)$	RSMT:$O(n^3)$ RSMT:$O(n)$			EMXT:$\theta(n\log n)$
Vor(S)	MAXG:$O(n\log n)$ MAXG:$O(n^3)$ ESPO: $O(n(k+\log n))$	ESPO: $O(n\log n)$		POFP: $O(n\log n, n, \log n)$	MINC:$O(n\log n)$
DT(S)	ESMTO:$O(?)$			RSMT:$O(n\log n)$ ESMT:$O(n\log n)$ ESMT:$O(n^{2.19})$ ESMT:$O(n^2)$	ANNP:$\theta(n\log n)$ EMST:$\theta(n\log n)$ RMST:$\theta(n\log n)$
VG(S)	ESPO:$O(n^2)$ ESPO:$O(k^2\log n,$ $n+k^2, n\log n+k^2)$ ESPO:$O(n\log n+$ $k^3, n+k^2, n\log n+$ $k^2)$ ESPO:$O(n^2\log n)$ ESPO: $O(n\log\log n)$ ESPO:$O(n^7L)$ ESPO:$O(n^3\log n)$ ESPO:$O(n^4\log n)$ ESPO:$O(n^2\log n)$ ESPO:$O(n^2)$ ESPO:$O(?)$ ESPO:$O(n^{k'})$ ESPO:$O(n^{a})$				
CD(S)	MAXG:$\theta(n)$ ETSP:$O_0(n\log n)$ ESMTO:$O(?)$ MAXG(Ω):$O(?)$		ANNP: $O(n(\log n)^{d-1})$ ETSP: $O_0(n\log n)$ RSMT: $O(n\log n)$	POFP: $O(n^{d+1}, \log n)$ POFP: $O(n^2, (\log n)^2)$	MAXG:$O_0(n^2)$ ANNP: $\theta(n\log n)$ EMST: $O((n\log n)^{1.5})$

续表

几何方法 / 几何结构	累　接	平面扫描	分治法	轨　迹	修剪与搜索
其他	MINC:$O(n^2)$ SMBO:$O(n^4)$ EMST: $O(n^2\log n)$ ETSP:$O(\exp)$ ESMT:$O(?)$ ESMT:$O(n^3)$ RSMT:$O(n^3)$ RSMT:$O(n\log n)$ ESMTO:$\theta(n)$	RSMT:$O(n\log n)$	CPP: $\theta(n\log n)$ EMST: $O(n^2\log n)$		MINC:$\theta(n)$ ETSP: $O(n^{2.5}(\log n)^4)$ ESMT:$O(\exp)$

第 10 章　随机几何算法与并行几何算法

在本章中，将 S 计算几何视为高维中的分类(sorting)和搜索(searching)问题来研究。在实轴 R 上给定 n 个点的集合 S，要求依据点的坐标将它们分类。

分类问题：寻找由给定点集 S 所形成的 R 的划分 $H(S)$。

形式上，划分 $H(S)$ 是由 S 中点确定的，而且 R 上的开区间由相邻的点构成。相关的搜索问题的几何表示如下：

搜索问题：建立与 $H(S)$ 相关的搜索结构 $\tilde{H}(S)$，使得对于给定的任意点 $q \in R$，耗费对数时间可以定位包含 q 的 $H(S)$ 中的区间。

搜索问题的动态形式是，通过联机方式增加或者删去一个点，从而改变集合 S，并且要求修改 $\tilde{H}(S)$ 只耗费对数时间。

高维中，集合 S 的元素依赖所考虑的问题，它或者是高维中的点、超平面或者是多边形。分类和搜索线性表的最简单的方法是随机方法，而几个高维分类和搜索问题的已知最简单的方法也是随机方法，因此可以把这些方法看作是分类和搜索线性表的随机方法的推广。下面首先介绍分类和搜索线性表的随机方法，并用几何语言重新阐述这些方法，然后推广到高维。值得注意的是 S 中的元素可以是数轴上的点，也可以是其他几何对象，只要它们可以线性地排序并且在常数时间内完成 S 中任意两个元素之间的比较。

本章 10.2、10.3、10.4 节分别介绍增量算法、动态算法和随机抽样，10.5 节叙述并行几何算法。

10.1　分类和搜索线性表的随机算法

分类实轴 R 上点的表的一个随机方法是快速分类，它是随机分治法的典型例子，其基本思想是从 R 中 n 个点的集合 S 中随机选取点 $p \in S$，它将 R 分成两个部分 S_1 与 S_2，然后递归地分类 S_1 与 S_2。如果 $|S_1| \approx |S_2|$，则递归的期望深度为 $O(\log n)$，因此算法的期望执行时间是 $O(n\log n)$。另外，利用随机数生成器产生的数(伪随机数)可以选择点 p。

放宽上述递归分类 S_1 与 S_2 的规定，可以得到随机增量的方法，它们将导致不同的算法。快速分类的随机增量方法的思想是，按随机顺序一次增加 S 中的一个点，也就是说，在任意给定时间，从未添加的点的集合中随机选取一个点 p，并把点 p 加入到现有的划分中。

设 S^i 是前 i 个添加点的集合，$H(S^i)$ 是由 S^i 所形成的 R 的划分，$H(S^0)$ 表示 R 的空划分，从 $H(S^0)$ 开始，构造划分序列

$$H(S^0), H(S^1), H(S^2), \cdots, H(S^n) = H(S)$$

执行算法的第 i 个阶段，对于每个区间 $I \in H(S^i)$ 还要保留其冲突表 $L(I)$。定义 $L(I)$ 是包含在 I 中的 $S\backslash S^i$ 内的点的未分类表；相反地，对于 $S\backslash S^i$ 中的每个点，保留指向包含它的

$H(S^i)$中冲突区间的指针。

从 $S\backslash S^i$ 中随机选取点 $s=S_{i+1}$进行加入，实际上是分割包含 s 的 $H(S^i)$中的区间以及它的冲突表。设 $I_1(s)$和 $I_2(s)$表示与 s 邻接的 $H(S^{i+1})$中的区间，如图 10-1 所示，$l(I_1(s))$和 $l(I_2(s))$表示它们的冲突规模，即相应冲突表 $L(I_1(s))$和 $L(I_2(s))$的规模。显然，加入 s 的耗费与 $l(I_1(s))+l(I_2(s))$成比例。

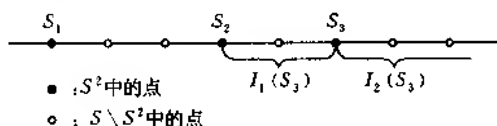


图 10-1　加入第 3 个点 S_3

随机几何算法与其他随机算法一样都具有一个重要的特征，即在算法中注入了随机性，而对输入集合中对象的概率分布不作任何假设，算法中的随机性保证了算法具有较低阶的期望复杂性。

估算快速分类的随机增量算法的时间期望复杂性如下：假设按随机顺序加入 S 中的点，由对称性所有可能的加入序列是等可能的。现计算第 $i+1$ 次加入的期望耗费，其思想是从 S^{i+1}到 S^i 反向进行，即删去 S^{i+1}中的随机点得到 S^i。已知加入 s 的耗费与 $l(I_1(s))+l(I_2(s))$成比例，这意味着第 $i+1$ 次加入的期望耗费(以确定的 S^{i+1}为条件)是

$$\frac{1}{i+1}\sum_{s\in S^{i+1}}[l(I_1(s))+l(I_2(s))+1]\leqslant\frac{2}{i+1}\sum_{J\in H(S^{i+1})}[l(J)+1]=O\left(\frac{n}{i+1}\right)$$

因为该限界不依赖于 S^{i+1}，所以它是第 $i+1$ 次加入的期望耗费的限界。因此算法的期望耗费为 $n\sum_i\frac{1}{i+1}=O(n\log n)$ 。

下面介绍随机二叉树(randomized binary tree)与跳越表(skip list)这两种数据结构，它们在随机几何算法设计、执行及分析中起重要的作用。

10.1.1　随机二叉树

随机二叉树与第 0 章介绍的线段树是类似的，叶结点与划分 $H(S)$中的区间一一对应，不同的是用随机选择的点 $s_1\in S$ 标记随机二叉树 $\hat{H}(S)$的根，点 s_1 分割 S 为两部分 S' 与 S''。定义根的左、右子树为递归定义的树 $\hat{H}(S')$和 $\hat{H}(S'')$，如图 10-2 所示。

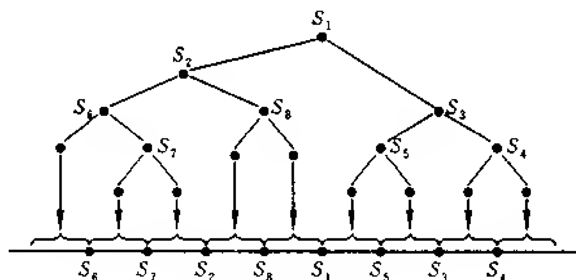


图 10-2　随机二叉树

快速分类的递归算法(分治算法)执行终止时,不仅得到划分 $H(S)$,而且还得到随机二叉树 $\tilde{H}(S)$,它可以用于定位 $H(S)$中的任意点 $q\in R$。定位点 q 的过程如下:将 q 与标记 $\tilde{H}(S)$的根的点 s_1 比较,如果 $q=s_1$,则任务完成了,即已定位点 q 为点 s_1。如果 q 的坐标小于(大于)点 s_1 的坐标,则在左(右)子树 $\tilde{H}(S')$($\tilde{H}(S'')$)上递归,最后将达到对应于包含 q 的 $H(S)$中区间的一个叶。

上述随机二叉树的定义可以修改为计算划分 $H(S)$的某个过程

$$H(S^0),H(S^1),H(S^2),\cdots,H(S^n)=H(S)$$

其中 S^i 是前 i 个随机选择的点的集合。通过对 i 的归纳,定义结构 $H(i)$为前 i 次加入的结构,然后定义 $\tilde{H}(S)$为 $H(n)$。$H(i)$的叶对应于 $H(S^i)$的区间,并用 S^i 中的点标记$H(i)$的内部结点。形式上,$H(0)$仅包含一个结点,它对应于整个 R。归纳假设已定义 $H(i)$,设 s_{i+1}是第 $i+1$ 个随机选取的点,I 是包含 s_{i+1}的 $H(S^i)$中的区间,s_{i+1}分割 I 成两个区间 I_1 和 I_2。第 $i+1$ 次加入期间删去区间 I,并产生新区间 I_1 和 $I_2\in H(S^{i+1})$。用 s_{i+1}标记$H(i)$的叶,它对应于 I,得到 $H(i+1)$,再给标记 s_{i+1}的叶赋两个子结点,它们分别对应于 I_1 和 I_2,如图 10-3 所示。

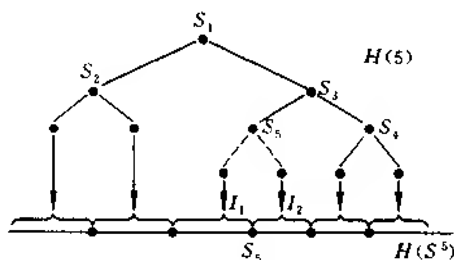


图 10-3 加入第 5 个点

显然,建立 $\tilde{H}(S)=H(n)$所需要的时间与快速分类的执行时间为同一数量级,其期望值为 $O(n\log n)$。

下面考虑搜索耗费(又称询问耗费或者点定位耗费),要求估算期望搜索耗费,所谓期望搜索耗费意指使用者在仅知道加入的顺序是随机的情况(不知道实际的加入顺序)下期望的搜索耗费。

固定询问点 $p\in R$,利用反向分析法,即从 $H(S)=H(S^n)$开始,依反向顺序 $s_n,s_{n-1},\cdots,s_1$,一次删去 S 中一个点。产生划分序列 $H(S^n),H(S^{n-1}),\cdots,H(S^0)$。对于 $1\leqslant j\leqslant n$,定义 0-1 随机变量 v_j,使得 $v_j=1$ 当且仅当包含 p 的 $H(S^j)$中的区间邻接于被删去的点 s_j。显然,$H(n)$中搜索路径的长度是 $V=\sum_{j=1}^{n}v_j$,并且 $v_j=1$ 的概率由$\frac{2}{j}$限界,因为 S^j 中至多有两个点与包含 p 的 $H(S^j)$中区间邻接。这意味着 v_j 的期望值亦以$\frac{2}{j}$限界。由期望值的线性性,推得 V 的期望值是 $O(\sum v_j)=O(\log n)$。这样对于固定询问点 p,搜索路径的期望长度是 $O(\log n)$。

可以证明,搜索路径的长度是 $\tilde{O}(\log n)$,即对数限界以高概率成立。这是依据调和随

机变量的 Chernoff 限界推得的。也就是说，搜索路径的长度大于 $\log n$ 的概率很小。其中符号 $\tilde{O}$ 的意义如下：如果对于某个正常数 c，$f(n)<cg(n)$ 以概率 $1-1/p(n)$ 成立，则称 $f(n)=\tilde{O}(g(n))$。其中 $p(n)$ 是其次数依赖于 c 的多项式。显然，$f(n)>cg(n)$ 以概率 $1/p(n)$ 成立。

如果去掉固定询问点的条件，那么搜索路径的期望长度仍然是 $\tilde{O}(\log n)$。因此得到下面的结论。

定理 10-1 给定数轴 R 上的 n 个点的点集 S，建立划分 $H(S)$ 及其相关的随机搜索树耗费 $\tilde{O}(n\log n)$ 时间，点定位耗费 $\tilde{O}(\log n)$ 时间。

上述是在已知点集 S 的情况下建立的搜索树 $\tilde{H}(S)$，该树是静态的。下面我们将使搜索结构动态化，其基本思想是简单的。如果 M 表示在任意时间 t 点的集合，那么 t 时的搜索结构看成是把静态方法应用于集合 M 而构造成的。这意味着 t 时的搜索结构的最大询问耗费是 $\tilde{O}(\log m)$，其中 $m=|M|$ 表示 t 时点的数目。

修改静态环境下的数据结构如下：对 M 中的每个点 s，从区间 $[0,1]$ 上随机选取一个实数（优先数）赋给点 s，M 中的点按此优先数排序，称为优先数顺序或者 Shuffle。M 中点的优先数是任意选取的，所以 M 上所有优先数排序是等可能的。我们用 Shuffle(M) 表示与 $H(M)$ 关联的数据结构状态，该状态由 M 的 Shuffle 确定。设 s_k 表示 M 上优先数顺序中的第 k 个点，M^i 表示前 i 个点 $s_1,s_2,\cdots,s_i$ 的集合。我们的数据结构是序列 $H(M^1),\cdots,H(M^m)=H(M)$ 的结构，换句话说，按点的优先数递增序添加 M 中的点，然后定义 Shuffle(M) 是该计算的结构。

Shuffle (M) 的另一定义是，用 M 中具有最低优先数的点标记 Shuffle(M) 的根，设 $M_1,M_2\subseteq M$ 分别是有较低和较高坐标的点的子集，根下面的左、右子树是递归定义的二叉树 Shuffle(M_1) 和 Shuffle(M_2)。

从 Shuffle(M) 中删去一个点的过程如下：设 $s\in M$ 是要删去的点，$M'=M\setminus\{s\}$，并且 M' 与 M 具有相同的优先顺序，因此亦定义了 Shuffle(M')。我们要求由 Shuffle(M) 推得 Shuffle(M')，分两种情况讨论：(1) 如果 s 在 Shuffle(M) 的底部（即叶），如图 10-3 中 S_5，s 的两个子结点是空的，也就是没有标记 M 中的点，此时删去这些子结点，如图 10-3 中删去标记 S_5 及其两个未标记的子结点。(2) s 不在 Shuffle (M) 的底部，此时，逐步增加 s 的优先数到最高可能值，从而将它引至树的底部。考虑用 s 标记的 Shuffle(M) 中的结点，如图 10-4 所示。设 s' 和 s'' 是标记 s 的结点的两个子结点的标记，不失一般性，设 s' 具有比 s'' 更低的优先数，其他情况是对称的。现设想增加 s 的优先数使之超过 s' 的优先数，这样，必须改变搜索树以适应优先数的变化。并因此要在树中 s' 的下面引入 s，因为 s 的新优先数更高。利用右旋转操作局部改变这棵树，如图 10-4 所示，得到的树正确反映 s 和 s' 的优先数顺序中的变动，其他情况与以前相同。用这种方式不断执行旋转，直至 s 沉到树的底部，然后删去 s。

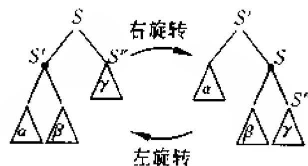


图 10-4 旋转

注意，删去操作终止时得到的 Shuffle(M') 就像它是通过把先前描述的静态方法应

用于集合 M' 所构造出的。

由于 Shuffle(M)的深度是 $\tilde{O}(\log m)$，并且每次旋转的耗费是 $O(1)$，因此删去操作的耗费是 $\tilde{O}(\log m)$。

给 Shuffle(M)增加一个新点 s 的加入操作如下：从区间[0,1]中为 s 以随机并独立于 M 中其他点的方式选择一个优先数，这便确定了集合 $M'=M\cup\{s\}$ 的优先顺序。目的是由 Shuffle(M)得到 Shuffle(M')。分两种情况讨论：(1) s 的优先数在 M 中是最高的。此时，定位对应于包含 s 的 $H(M)$ 中区间的 Shuffle(M)的叶，用 s 标记这个结点，并给它两个子结点(这些子结点对应于与 s 邻接的 $H(M')$ 中的两个区间)。由于搜索路径的长度是 $\tilde{O}(\log m)$，所以加入操作需要 $\tilde{O}(\log m)$ 时间。(2) s 的优先数不是 M' 中最高的。在搜索树中通过旋转将 s 置于适当位置，这个操作是删去期间类似操作的反向。因此，它的耗费有相同的阶。

总之，给 Shuffle(M)加入 s 的耗费是 $\tilde{O}(\log m)$。注意，加入操作终止时所得到的 Shuffle(M')就像它是把先前描述的静态方法应用于集合 M' 所构造的。

10.1.2 跳越表

本段描述分类表中动态搜索的另一随机结构，称为跳越表。首先，阐述静态环境下的搜索结构，然后将它动态化。设 M 是实轴 R 上 m 个点的集合，利用底-向上的随机抽样技术(见 10.4.3 节)，把搜索结构与 M 联系起来，并用 Sample(M)表示搜索结构。

从集合 M 开始，得到如下的集合序列

$$M=M_1\supseteq M_2\supseteq\cdots\supseteq M_{r-1}\supset M_r=\varnothing$$

其中 M_{i+1} 是由 M_i 用下述方法得到：对于 M_i 中每个点独立地翻动一枚硬币，并且仅保留抛硬币获得成功(正面向上)的那些点。上述集合序列称为 M 的一个分层(gradation)。注意，每个 M_i 的规模大概是 M_{i-1} 规模的一半，因此该分层的期望长度是 $O(\log m)$。搜索结构 Sample(M)的状态完全由 M 的分层来确定，这一点类似于 Shuffle(M)的状态完全由 M 的优先数顺序确定。

Sample(M)由 r 级组成，其中 r 是上述分层的长度，$r=O(\log m)$。集合 $M_i(1\leqslant i\leqslant r)$ 可以看成是第 i 级中所存储的点的集合，如图 10-5 所示。我们将用 Sample(M)的第 i 级的分类链接表的形式存储划分 $H(M_i)$，这意指必须分类 M_i(在这种表示中，$H(M_i)$ 的区间是隐含的)。另外，还把指向第 $i-1$ 级中相同点的下降指针同存储在第 i 级的点 $s\in M_i$ 联系起来。为简单起见，假设每个 M_i 包含坐标 $-\infty$ 的额外点。

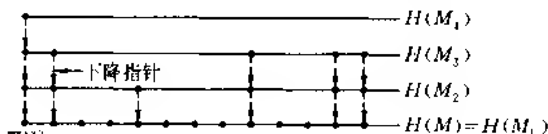


图 10-5 跳越表

所谓 Sample(M)中的区间是指 Sample(M)的任何级中的区间。给定区间 $I\in H(M_i)$ 的父区间定义为包含 I 的 $H(M_{i+1})$ 中的区间 J；亦称 I 是 J 的一个子区间。如果设想从父

区间到子区间有指针，那么跳越表可以看成是搜索树。

Sample(M)可以用于搜索，其过程如下：设 s 是要在 $H(M)$中定位的点。首先，平凡地定位 s 在 $H(M_r)$中，它仅由一个区间组成，即整个 R。归纳假设 s 已定位在 $H(M_i)$，即 s 定位于存储在第 i 级($1<i\leqslant r$)的划分中。设$\triangle_i \in H(M_i)$是包含 s 的区间，利用与$\triangle_i$ 的左端点相关的下降指针可以搜索$\triangle_i$ 的所有子区间，便得到包含 s 的第 $i-1$ 级中所需要的区间$\triangle_{i-1}$。当搜索达到第一级时，便完成了搜索，如图 10-6 所示。注意，搜索耗费与搜索路径上子区间数成正比，所谓搜索路径上的区间是指包含 s 的 Sample(M)的所有级中的区间。

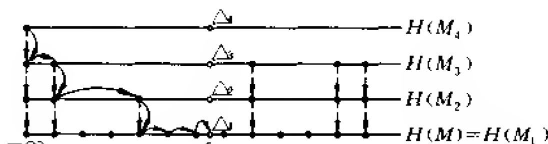


图 10-6　搜索路径

可以证明 Sample(M)中的期望级数以高概率为 $\tilde{O}(\log m)$，构造 Sample(M)耗费 $\tilde{O}(m\log m)$时间，而期望搜索耗费是 $O(\log m)$。

新点 s 加入 Sample(M)的过程如下：反复抛一枚硬币直到失败(反面向上)，设 j 是失败之前成功的次数，目的是通过 $j+1$ 把 s 加入到 1 级。对于每个 i，Sample(M)中 s 的搜索路径报告包含 s 的区间$\triangle_i \in H(M_i)$，如图 10-6 所示。因此，对于每个 i，$1\leqslant i\leqslant j+1$，只需要分割每个$\triangle_i$，并用下降指针连接相继级中出现的 s，如图 10-7 所示。当 $j+1$ 大于当前深度 r 时，需要产生仅包含 s 的额外 $j+1-r$ 级。上述过程的耗费是 $\tilde{O}(\log m)$。

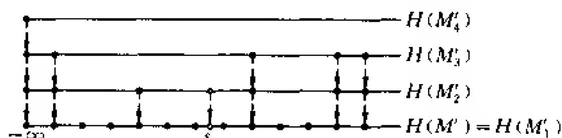


图 10-7　加入新点 s

设 $M'=M\cup\{s\}$。对于 $1\leqslant l\leqslant j+1$，$M_l'$ 表示 $M_l\cup\{s\}$；并且对于 $l>j+1$，令 $M_l'=M_l$。上述加入过程终止时，得到已修改的数据结构 Sample(M')，它对应于分层

$$M' = M_1' \supseteq M_2' \supseteq \cdots$$

看上去该分层好像是将本段开始的静态过程应用于集合 M' 而构造的。

删去点 $s\in M$ 时，只要由包含 s 的每级中删去它，每级删去的耗费为 $O(1)$。包含 s 的期望级数是 $O(1)$，因此，删去 s 的期望耗费为 $O(1)$。

10.2　增量算法

S 计算几何中高维分类问题描述如下：给定 E^d 中的对象集合 S，快速构造递归的几何划分 $H(S)$。对象集合 S 和划分 $H(S)$都依赖于所考虑的问题。

构造 $H(S)$的最简单方法是增量算法。所谓增量算法是指一次增加一个对象逐步构

造 $H(S)$的算法。前面阐述的平面凸壳以及线段排列等均已使用了增量算法。由于这种方法不能有效地工作，所以对其进行改进，即按随机顺序一次增加一个对象，改进后的算法称为随机增量算法。随机增量算法的抽象形式如下：

步 1 构造 S 中对象的随机序列（置换）$s_1, s_2, \cdots, s_n$。

步 2 依据该随机序列的顺序一次增加一个 S 中的对象。设 S^i 表示前 i 个增加对象的集合。在第 i 阶段，算法保持由集合 S^i 形成的划分 $H(S^i)$及相关的辅助数据结构。每次增加一个对象时，$H(S^i)$与辅助数据结构都要进行有效地修改。

增量算法不一定都是联机（半动态）的，但本节中的增量算法可以转换成联机算法。

联机算法（半动态算法）的任务是要有效地保持划分 $H(M)$（其中 M 表示在任意时刻已增加对象的集合），并且新增对象之后要快速修改 $H(M)$。该修改所需要的时间与修改过程中划分 $H(M)$的结构变化成比例。所谓修改过程中的结构变化是指在该过程中新产生的及删去的小面的总数。例如考虑平面上线段排列问题，此时 M 是平面上线段的集合，$H(M)$是所得到的排列，任意增加过程中结构变化大约是 m（M 的规模）。

一般来说，随机增量算法有较好的期望时间复杂性。下面给出增量算法的几个实例。

10.2.1 四边形分解

给定平面上 n 条线段的集合 S，如图 10-8(a)所示。过 S 中线段端点及交点作垂线，这些垂线延伸至首次碰到 S 中一条线段为止，便得到平面的四边形分解，记为 $H(S)$，如图 10-8(b)所示。$H(S)$的规模是 $O(k+n)$，其中 k 是 S 中相交线段的数目。本段将给出构造 $H(S)$的随机增量算法，该算法的时间复杂性为 $O(k+n\log n)$，低于扫描算法的时间复杂性 $O(k\log n+n\log n)$，但以损失确定性为代价。

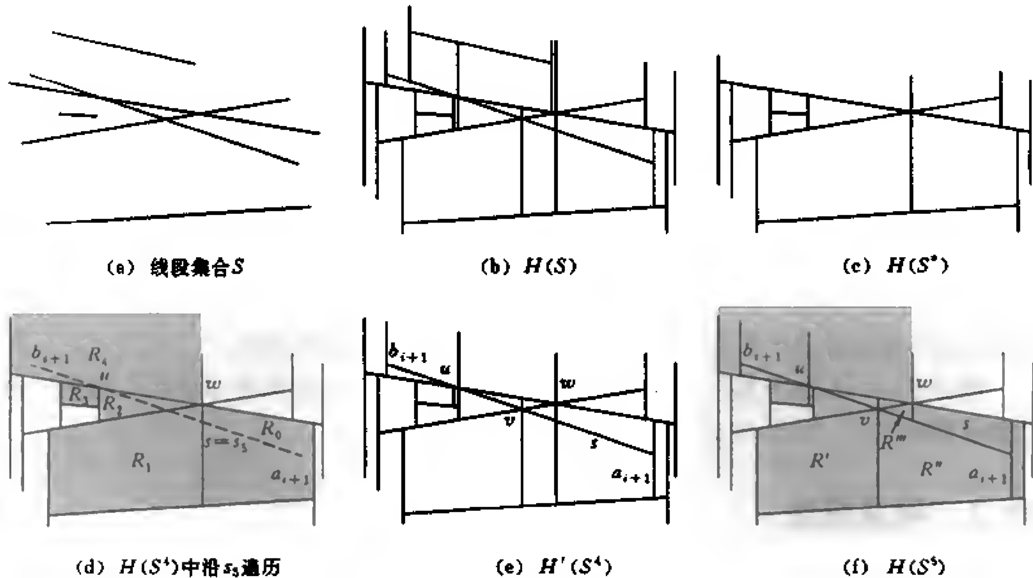


图 10-8 平面的四边形分解

开始时，$H(S)$是空集。然后按随机顺序一次选取 S 中一条线段加入到 $H(S)$。算法执

行的第 i 阶段，要构造 $H(S^i)$，其中 S^i 表示前 i 条随机选取线段的集合，如图 10-8(c)所示。第 $i+1$ 次随机选取的线段记为 $s=s_{i+1}$(a_{i+1}和 b_{i+1}表示它的两个端点)，用下述方法将 s_{i+1}加入到 $H(S^i)$：

步 1 遍历和分裂。假设已知包含 a_{i+1}的四边形 $R_0 \in H(S^i)$，从 a_{i+1}出发遍历 s 到达 b_{i+1}。在图 10-8(d)中，该遍历过程要碰到 $H(S^i)$中的小面 R_0，R_1，R_2，R_3 和 R_4。沿 s 穿过小面 f 所需要的时间与面长度(f)成比例，其中面长度(f)表示与 f 相邻的 $H(S^i)$的顶点数。对 s 穿过的小面 f 需要进行分裂处理：如果 s 与 f 的上侧面或下侧面交于点 v，则过点 v 作垂线 l，延伸 l 直至首次碰到 $H(S^i)$中的线段。如果 s 中的端点位于 f 的内部(比如图 10-8(d)中的 R_0 与 R_4)，则过端点作垂线，并延伸垂线至首次碰到 $H(S^i)$中的线段。显然，小面 f 至多被分裂成 4 个更小的面，并且分裂工作可以在与面长度(f)成比例的时间内完成。本步完成之后得到划分 $H'(S^i)$，如图 10-8(e)所示。

步 2 收缩与合并。如果过 $H(S^i)$中交点 w 的垂直线与 s 相交(设交点为 w')，则收缩该垂直线使终止于 s 上，如图 10-8(f)所示。任意垂直线的收缩需要 $O(1)$时间。收缩之后，便合并与该垂直线删去部分相邻的面(如图 10-8(f)中的 R'')。

由上面的讨论得到下述的结果。

命题 10-1 如果已知包含 $s=s_{i+1}$的一个端点的 $H(S^i)$中的四边形，则在与 $\sum_f$ 面长度(f)成比例的时间内可以把 $H(S^i)$修改为 $H(S^{i+1})$，其中小面 f 是指与 s 相交的 $H(S^i)$中的所有四边形。

算法执行的第 i 阶段保持一个无序的冲突表 $L(f)$，并且每个小面 f 在 $H(S^i)$中。$L(f)$包含未增加线段的端点(位于 f 内部)，由与 s 的每个端点有关的冲突指针可以获得包含端点的 $H(S^i)$中的四边形，而且这仅需 $O(1)$时间。另一方面，增加 s 时，除了分裂和合并面之外，还要将它们的冲突表作相应的分裂和合并。冲突表的分裂在与其规模成比例的时间内完成：如果小面 f 被分裂成两个更小的面 f_1 和 f_2，那么对于 f 的冲突表中的每个点要确定它位于 f_1 中还是 f_2 中。通过连接可以合并冲突表，每次合并需要 $O(1)$时间。

命题 10-2 修改冲突表的耗费是 $O(\sum_f l(f))$，其中 f 是与 s 相交的 $H(S^i)$中的所有四边形，而 $l(f)$表示 f 的冲突规模，即冲突表 $L(f)$的规模。

由命题 10-1 和 10-2 可以得到，把 s_{i+1}加入到 $H(S^i)$的总耗费是 $O(\sum_f$ 面长度$(f)+l(f))$，其中 f 是与 s 相交的 $H(S^i)$中的所有四边形。利用反向推理方法(即从 $H(S^{i+1})$中删去 s_{i+1})可以得到下面的结果(省略证明的细节)。

定理 10-2 平面上 n 条线段所构成的四边形分解可以在 $O(k+n\log n)$期望时间内构成，其中 k 表示相交线段的数目。

上述算法是增量的而不是联机的，这是因为它保持冲突表，在算法的第 i 个阶段它依赖于 $S \backslash S^i$ 中的线段。冲突表有助于在 $H(S^i)$中确定 s_{i+1}的一个端点。如果能建立与$H(S^i)$有关的联机搜索结构，并用 *History*(i)表示这个搜索结构(简记为 $H(i)$)，那么就没有必要保持冲突表。这样，便将上述增量算法转换成联机的。

$H(i)$表示的搜索结构类似于二叉搜索树。算法执行的第 i 个阶段保持 $H(S^i)$及搜索结构 $H(i)$ 。$H(i)$的叶指向 $H(S^i)$中的四边形。对于给定的任意查询点 p，可以在$O(\log i)$

时间内以高概率定位包含 p 的 $H(S^i)$中的四边形。这样便将上述的随机增量算法转变成联机算法。该联机算法进行点定位的期望总耗费是 $O(\sum_i \log i) = O(n\log n)$，因此，算法的总期望时间是 $O(n\log n+k)$，其中 k 是相交线段的总数目。

可以递归地确定 $H(i)$。首先，$H(0)$由一个结点(称为根)组成，它对应于整个平面。然后归纳假设已经确定 $H(i)$。由 $H(i)$得到 $H(i+1)$的过程如下：设 $s=s_{i+1}$表示要加入的第 $i+1$ 条线段。如果 $H(S^i)$中的四边形 f 与 s 相交，则加入 s 期间 f 被分裂。因此，当被线段 s 分裂时标记 $H(i)$中指向 f 的叶，并把该叶与指向 s 的指针联系起来。图 10-8(d)中，加入 s_5 时，四边形 R_0、R_1、R_2、R_3 和 R_4 被分裂，标记对应于 R_0、R_1、R_2、R_3 和 R_4 的 $H(4)$的叶。

类似地，在第 $i+1$ 次加入期间定位新近产生的 $H(S^{i+1})$中四边形的新结点，对应结点处存储新产生的四边形。因此，图 10-8(d)中阴影域定位新结点。$H(S^i)$中给定一个分裂的四边形 f 及 $H(S^{i+1})$中新产生的四边形 g，如果 $f\cap g\neq\varnothing$，则 g 是 f 的一个子结点。在这种情况下，把指针与对应于 f 的 $H(i)$中的结点联系起来，其中 f 指向 g 的新产生的结点(叶)，这就由 $H(i)$确定了 $H(i+1)$，如图 10-8(d)～(f)所示。$R_1\in H(S^4)$的子结点是 $R',R'',R'''\in H(S^5)$。注意，$H(i)$中任意结点的子结点数目不超过 4。

$H(i)$是一个有根的非循环直线图，其中直线边对应于从父结点到子结点的指针，每个结点的出度至多是 4。从 $H(i)$的根开始，根的一个子结点对应于包含查询点 p 的一个四边形，因为子结点的数目至多是 4，所以可以在 $O(1)$时间内确定该子结点。然后继续该过程，沿 $H(i)$中某条路径向下搜索，直至达到一个叶，该叶对应于包含 p 的 $H(S^i)$中的四边形。

显然，定位询问点 p 的耗费与 $H(i)$中的搜索路径的长度成比例，有下面的结论。

引理 10-1 对于一个确定的查询点，$H(i)$中的搜索路径的长度是 $\tilde{O}(\log i)$。

证明 采用随机二叉树及与加入过程相反的顺序来证明。设 p 是一个确定的查询点，想象依据反向顺序 $s_i,s_{i-1},\cdots,s_1$ 从 S^i 中一次删去一条线段。对于 $1\leqslant j\leqslant i$，定义一个 0-1随机变量 V_j 使得 $V_j=1$ 当且仅当包含 p 的 $H(S^j)$中的四边形与 s_j 邻接。注意，由 S^j 删去 s_j 的过程中该四边形将消失。比如，如果 p 位于图 10-8(f)中任意阴影四边形中，那么 V_5 将是 1。显然，$H(i)$中搜索路径的长度是 $\sum_{j=1}^{i} V_j$。但 S^j 中的每条线段可能像 s_j 那样等概率出现，因此，$V_j=1$ 的概率$\leqslant\frac{4}{j}$。这是因为 S^j 中至多有 4 条线段与包含 p 的 $H(S^j)$中的四边形邻接。这意味着 V_j 的期望值是 $O(1/j)$。由期望值的线性性，得出搜索路径的期望长度是 $O\left(\sum\frac{1}{j}\right)=O(\log n)$。再利用 Chernoff 技术，便得到引理的结论成立。

证毕。

引理 10-2 对于任意 $k\leqslant i$，如果已知对应于包含查询点 p 的 $H(S^k)$中四边形的 $H(k)$的结点，那么可以在 $O\left(1+\log\frac{i}{k}\right)$期望时间内定位包含 p 的 $H(S^i)$中的四边形。

证明 连接结点的搜索路径长度是 $\sum_{j=k}^{i} V_j$，它对应于包含 p 的 $H(S^k)$和 $H(S^i)$中的

四边形，因此期望值是 $O\left(\sum_{j=k}^{i}\frac{1}{j}\right)=O\left(\log\frac{i}{k}\right)$。 证毕。

有界度性质：$H(S^i)$上每个询问的回答是由 S^i 中对象的有界数（指由一个常数限界的数）确定的。

当前，$H(S^i)$上的询问的回答是包含询问点 p 的四边形，显然，它是由 S^i 中与四边形邻接的线段的有界数确定的。一般情况下，只要有界度性质成立，引理 10-1 就成立。

引理 10-1 证明了关于查询点的搜索路径长度的高概率限界。而 $H(i)$中搜索路径总数受囿于 i 中固定度数的多项式，即 $O(i^2)$。$H(i)$的深度是 $\tilde{O}(\log i)$。

如果 S 中线段除了在其端点相交外，它们不相交，那么 S 中的线段便构成一个平面图。可以用上述算法来构造该平面图的四边形分解以及关于该四边形分解的点定位结构。这个结构可以用来定位包含一个给定询问点的四边形，一旦知道包含询问点的四边形，也就知道包含该点的平面图的小面。这样便可得到关于平面图的点定位结构。

如果 S 中线段由一个简单多边形的边组成，那么四边形分解算法的时间复杂性是 $O(n\log n)$，这是因为在这种情况下没有线段相交，即 $k=0$。四边形分解算法仍然按随机顺序一次加入一条线段，并保持 $H(S^i)$及 $H(i)$。如果不计定位加入线段的一个端点的耗费，那么算法的期望时间是 $O(n)$。联机四边形分解算法时间复杂性中的 $O(n\log n)$项是由于点定位耗费而产生的，只要改进定位端点的方法，便可降低算法复杂性。现有的结果是期望时间复杂性为 $O(n\log^* n)$，如果从数 n 开始并以小于 1 的数终止，那么 $\log^* n$ 表示需要执行对数运算的数目。也就是说，它是使 n 的第 k 次迭代对数小于 1 的最小的正整数 k。

定理 10 3 简单多边形的四边形分解及点定位结构可以在 $O(n\log^* n)$期望时间内构成，其中 n 是简单多边形顶点的数目。点定位的耗费是 $\tilde{O}(\log n)$。

10.2.2 凸多胞形

设 S 是 E^d 中 n 个半空间的集合，其中 $d=2$ 或者 3。另设 $H(S)$是由这些半空间的交形成的凸多胞形，$d=2$ 时，$H(S)$恰是一个凸多边形。本段先给出一个构造 $H(S)$的随机增量算法，它的期望时间复杂性为 $O(n\log n)$，然后说明如何将该算法转换成联机算法。

设 S^i 表示随机增加的前 i 个半空间的集合，$H(S^i)$是 S^i 对空间的划分，并设 $H(S^i)$是有界的。为了确保 $H(S^i)$的有界性，只要把 S 限制在一个充分大的超矩形域内。如果已经构造了 $H(S^i)$，那么由 $H(S^i)$构造 $H(S^{i+1})$，即向 $H(S^i)$加入第 $i+1$ 个半空间 $s=s_{i+1}$的步骤如下：

步 1 随机选取一半空间作为 $s=s_{i+1}$。

步 2 在 $H(S^i)\cap\bar{s}$ 中选择 $H(S^i)$的一个顶点，记为 p，其中 $\bar{s}$ 表示 s 的补（称 p 为与 s 冲突的顶点），转步 4。

步 3 如果 $\bar{s}$ 中不存在冲突顶点 p，则删去 s，输出 $H(S^{i+1})=H(S^i)$。

步 4 由 p 出发，沿 $H(S^i)$的边搜索到 s，由搜索过的边和 s 组成连通域 $H(S^i)\cap\bar{s}$，记为 $\text{cap}(s_{i+1})$。

步 5 $H(S^{i+1})=H(S^i)\backslash\text{cap}(s_{i+1})$。

如果用 f 表示交于 $\bar{s}$ 的 $H(S^i)$的 2-面，并且 f 完全位于 $\bar{s}$ 的内部，则删去 f 和相邻的

边及顶点。如果 f 部分地位于 $\bar{s}$ 的内部，则分割 f。也就是说，用 $f\cap s$ 代替 f。删去位于 $\bar{s}$ 内 f 的边。分割部分交于 $\bar{s}$ 的 f 的两条边。产生一条新边 $f\cap\partial s$，其中 ∂s 表示 s 的边界。最后，还要产生新的面 $H(S^i)\cap\partial s$，修改相关信息，这就完成了由 $H(S^i)$ 构造 $H(S^{i+1})$ 的任务。如图 10-9 所示。

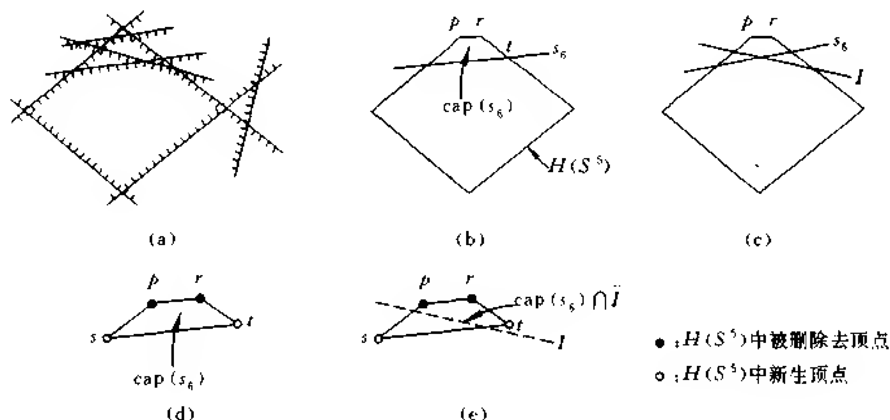


图 10-9　构造凸多胞形

(a) 半空间集合 S　(b) 修改 $H(S^5)$或 $H(S^6)$　(c) $H(S^5)$中的冲突　(d) cap(s_6)　(e) 冲突的重新定位

如果不计寻找冲突点 p 的耗费，那么加入 $s=s_{i+1}$ 的耗费与 $H(S^i)$ 中被删去的顶点数加上 $H(S^{i+1})$ 中新产生的顶点数成比例，或者与包含在 ∂s_{i+1} 中的 $H(S^{i+1})$ 的顶点数(记为 $m(s_{i+1},S^{i+1})$)成比例。

利用反向推理方法，设想从 S^{i+1} 中删去 s_{i+1}。由于 S^{i+1} 中的每个半空间可以等概率出现，因此期望耗费与下述等式成比例

$$\frac{1}{i+1}\sum_{s\in S^{i+1}} m(s,S^{i+1})$$

其中和是由 d 乘上 $H(S^{i+1})$ 的顶点数所限界，得到结论：

引理 10-3　对于给定的 S^{i+1}，第 $i+1$ 次随机加入时新产生的顶点期望数由 $\frac{d}{i+1}$ 乘以 $H(S^{i+1})$ 的顶点数所限界。

因为 S^{i+1} 是 S 的随机抽样，没有任何约束条件下的期望值是由 $de(i+1)/(i+1)$ 所限界，其中 $e(i+1)$ 表示 $i+1$ 时 $H(S^{i+1})$ 的顶点数的期望数。$d=2,3$ 时，该期望数是 $O(i+1)$。因此，如果给定一个冲突点 p，那么第 $i+1$ 次加入的期望耗费是 $O(1)$。

下面说明如何定位冲突点 p。基本想法与上一段中介绍的四边形分解算法的思想类似。算法执行的第 $i+1$ 阶段，对于半空间 $I\in S\backslash S^i$ 保持指向与其冲突的 $H(S^i)$ 中的一个顶点的指针，设该顶点为 r，见图 10-9(c)～(e)。r 是 cap(s_{i+1}) 中的一个顶点。我们需要寻找与 I 冲突的 $H(S^{i+1})$ 的另一个顶点。此时，包含在 ∂s_{i+1} 中的一个顶点必与 I 冲突。图 10-9(e) 中顶点 t 即为所求的顶点。

$d=3$ 时，$H(S^i)$ 的顶点包含在三个限界平面中，而 $d=2$ 时，包含在两条限界直线中。因此，在最坏情况下，上述搜索与 cap$(s_{i+1})\cap\bar{I}$ 中的顶点数成比例，该顶点数等于与 I 冲

突的 $H(S^i)$ 中被删去的顶点数，加上与 I 冲突的 $H(S^{i+1})$ 中新产生的顶点数。每个新生顶点仅可被删去一次，因此可以忽略被删去的顶点数，这样搜索耗费与和 I 冲突的 $H(S^{i+1})$ 中新生顶点数 $k(S^{i+1},s_{i+1},I)$ 成比例，这些新生顶点在 ∂s_{i+1} 中，它们也是 $H(S^{i+1})$ 中与 S^{i+1} 邻接的冲突顶点。

定理 10-4 给定 E^d 中 $(d=2,3)n$ 个半空间的集合 S，由它们的交形成的凸多胞形可以在期望时间 $O(n\log n)$ 内构造。

证明 假设已给定 S^{i+1}，估计 $k(S^{i+1},s_{i+1},I)$ 的期望值。从 S^{i+1} 中随机选取 s_{i+1}，也就是说，s_{i+1} 以等概率从 S^{i+1} 中抽取。$H(S^{i+1})$ 中每个顶点与 S^{i+1} 中 d 个半空间邻接。因此，该期望值与下式成比例

$$\frac{1}{i+1}\sum_{s\in S^{i+1}} k(S^{i+1},s,I)\leqslant\frac{dk(S^{i+1},I)}{i+1}$$

其中 $k(S^{i+1},I)$ 表示与 I 冲突的 $H(S^{i+1})$ 中顶点的数目。对于给定的 S^{i+1}，第 $i+1$ 次加入时新产生顶点的期望冲突规模由下式限界

$$\frac{d}{i+1}\sum_{I\in N\backslash N^{i+1}} k(S^{i+1},I) \tag{10-1}$$

另一方面，假设 s_{i+2} 等概率地从 $S\backslash S^{i+1}$ 中抽取，所以 $k(S^{i+1},s_{i+2})$ 的期望值是

$$E[k(S^{i+1},s_{i+2})]=\frac{1}{n-(i+1)}\sum_{I\in S\backslash S^{i+1}} k(S^{i+1},I)$$

因此，式(10-1)可以写成

$$d\,\frac{n-(i+1)}{i+1}E[k(S^{i+1},s_{i+2})]$$

但是也可以把 $k(S^{i+1},s_{i+2})$ 解释为第 $i+2$ 次加入期间被删去的 $H(S^{i+1})$ 的顶点数。对 $i+1$ 求和，得到算法执行中所产生的全部顶点的期望总冲突规模，它由下式限界

$$\sum_{i=0}^{n-1} d\,\frac{n-(i+1)}{i+1}\times(i+2)\text{ 时删去顶点的期望数} \tag{10-2}$$

算法执行中产生的所有顶点的期望总冲突规模也由下式限界

$$\sum_{i=0}^{n-1} d\,\frac{n-(i+1)}{i+1}\times(i+1)\text{ 时产生顶点的期望数} \tag{10-3}$$

由引理 10-3，式(10-3)可以表示为

$$\sum_{i=0}^{n-1} d^2\,\frac{n-(i+1)}{i+1}\,\frac{e(i+1)}{i+1} \tag{10-4}$$

其中 $e(i+1)$ 表示在时间 $i+1$ 凸多胞形 $H(S^{i+1})$ 上顶点的期望数。$d=2,3$ 时，该顶点数是 $O(i)$。所以 $e(i+1)$ 是 $O(i)$。因此，算法执行的期望耗费为 $O\left(\sum_i\frac{n-(i+1)}{i+1}\right)=O(n\log n)$。

证毕。

上述算法是增量的但不是联机的，这是因为算法保持未加入半空间的冲突，这些冲突在加入 s_{i+1} 期间提供 $H(S^i)$ 的一个顶点位于 $\bar{s}_{i+1}$ 中。与上段类似，引入联机搜索结构 $H(i)$，可以使算法变成联机随机增量算法。

假设 S^0 包含的半空间是一个充分大的矩形，因此，$H(0)$ 包含对应于该矩形的角的结点。归纳假设已确定 $H(i)$，由 $H(i)$ 构造 $H(i+1)$ 的过程如下：在时间 $i+1$ 时，如果 $H(S^i)$

中一个顶点与 s_{i+1} 冲突，则删去该顶点。因此，当由半空间 s_{i+1} 删去时在 $H(i)$ 中标记相应的叶。还要定位 $H(S^{i+1})$ 中新生顶点的 $H(i+1)$ 中的新结点。这些新生结点与cap(s_{i+1})的底部顶点一一对应。类似地，$H(i)$ 中被删去的结点与 cap(s_{i+1})中剩余顶点一一对应。因此，连接这些被删去的和新生的顶点构成链 $l(s_{i+1})$，它是 cap(s_{i+1})的边界，如图 10-9(d)所示。

给定半空间 $I \notin S^i$，利用结构 $H(i)$ 寻找与 I 冲突的 $H(S^i)$ 的一个顶点的过程如下：

步 1 在结构 H 中寻找结点 t，它对应于与 I 冲突的 $H(S^0)$ 的一个顶点，耗费 $O(1)$ 时间。

步 2 设 $s_j(j \leqslant i)$ 是删去 t 的半空间，利用链 $l(s_j)$ 寻找对应于与 I 冲突的 $H(S^j)$ 中顶点的结点 t'，$l(s_j)$ 对应于 cap(s_j)的边界。

步 3 搜索 cap(s_j)的边界。

步 4 重复步 2、步 3，直至达到对应于与 I 冲突的 $H(S^i)$ 中一个顶点的 $H(i)$ 的一个叶时终止。

上述过程的第 i 阶段保持 $H(S^i)$ 及 $H(i)$，利用结构 $H(i)$ 可以定位与 s_{i+1} 冲突的 $H(S^i)$ 的顶点。这个过程将凸多胞形的随机增量算法变成随机联机算法。但该过程不改变原算法复杂性的量级，只是增加常数倍。也就是说，构造凸多胞形的随机联机算法的期望时间复杂性仍然是 $O(n\log n)$。

10.2.3 Voronoi 图

三维凸多胞形投影到平面便得到平面上的 Voronoi 图，因此上一段中介绍的算法可以产生一个构造平面上 Voronoi 图的随机联机算法，该算法的期望时间是 $O(n\log n)$，其中 n 为点集中点的数目。

下面考虑 Voronoi 图中点定位问题，也就是要构造 Voronoi 图的联机点定位结构。

设 S 是平面上 n 个点的集合，Vor(S)是 S 中点形成的 Voronoi 图。上一段中随机联机算法直接处理 Voronoi 图的过程如下：依据随机顺序每次增加一个点。设 S^i 表示前 i 个随机加入点的集合，在每个阶段 i，算法保持 Vor(S^i)及上述加入的结构 $H(i)$。由Vor(S^i)构造 Vor(S^{i+1})的过程如图 10-10(a)、(b)所示，设 $S=s_{i+1}$ 表示第 $i+1$ 次加入的点，步骤如下：

步 1 定位与 $S=s_{i+1}$ 冲突的点 p，p 是位于 S 所在的 Voronoi 域中的 Vor(S^i)顶点。可能有多个 Vor(S^i)顶点与 s 冲突，这些冲突顶点和相邻边构成一个连通图，图 10-10(b)中用点线表示该连通图。

步 2 计算 Vor(s)的边，s 的 Voronoi 域。

步 3 删去与 s 冲突的所有点及相邻边。

阶段 i 的结构包含关联于每个顶点的唯一结点，该顶点作为某个 Vor(S^j)($j \leqslant i$)顶点产生。第 $i+1$ 次加入期间所产生的连接结构 $l(s_{i+1})$ 对应于 Vor(s_{i+1})的边界上的新顶点及 Vor(s_{i+1})内的顶点之间的连接边，见图 10-10(c)。

用于搜索的结构如同凸多胞形情况。设给定一点 $I \notin S^i$，定位与 I 冲突的 Vor(S^i)的一个顶点如下：首先寻找对应于与 I 冲突的 Vor(S^0)顶点的结构中的一个结点（设

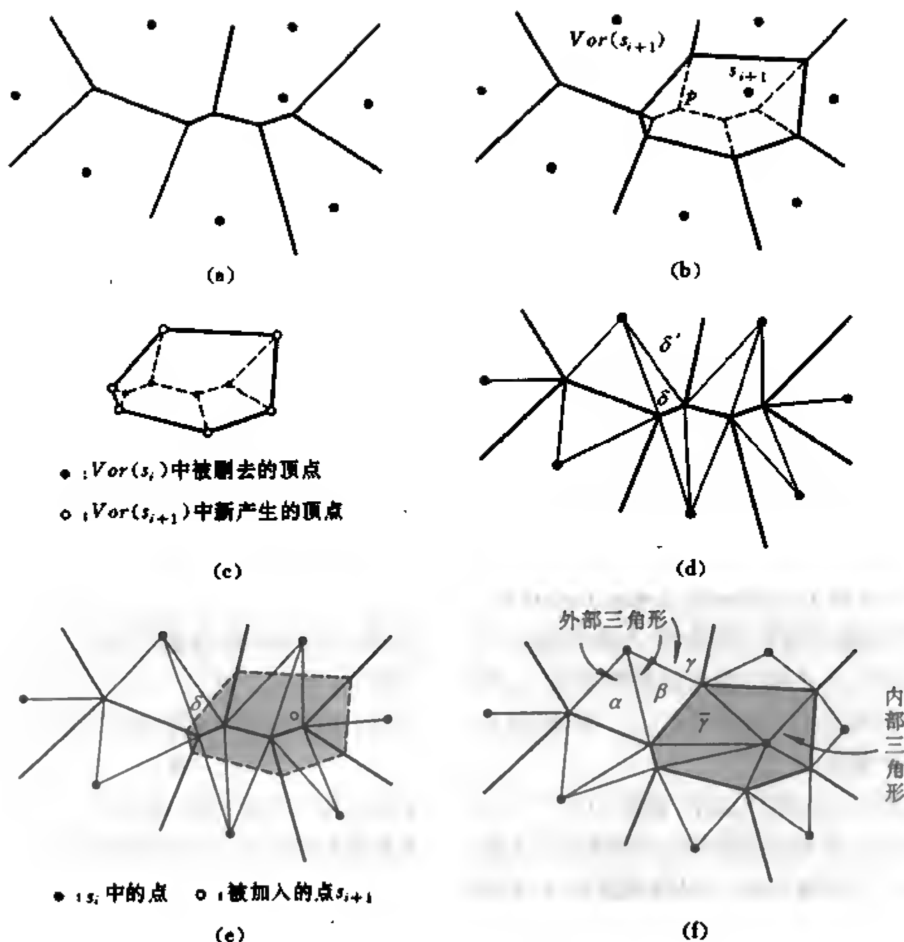


图 10-10　构造 Voronoi 图

(a) $\mathrm{Vor}(S^i)$　(b) $\mathrm{Vor}(S^{i+1})$　(c) $I(s_{i+1})$

(d) $H(S^i)$　(e) s_{i+1}加入到 $H(S^i)$　(f) $H(S^{i+1})$

$\mathrm{Vor}(S^0)$中单个无穷域是一个近似无穷的超立方体)。归纳假设在与 I 冲突的结构中某个结点 t，假设在阶段 j 删去 t。利用连接 $l(s_j)$可以寻找对应于与 I 冲突的 $\mathrm{Vor}(S^j)$中顶点的一个结点，其过程如下：首先，与 I 冲突的 $l(s_j)$的结点构成它的连通子图(与 10.2.2 小节中 $\mathrm{cap}(s_{i+1})\cap\bar{I}$ 类似)。通过搜索，利用连通性可以定位对应于 $H(S^j)$中一个冲突顶点的一个结点。从 t 开始并在上述 $l(s_j)$的连通子图内搜索。一旦找到对应于 $\mathrm{Vor}(S^j)$中的冲突顶点的结点 σ，就停止搜索。然后用类似方法在结构中由 σ 向下搜索，不断重复此过程直至在阶段 i 达到结构的叶为止。

上述构造 Voronoi 图的随机联机算法的描述只是凸多胞形的随机联机算法的一个转换，因此有下面的结论。

定理 10-5　以联机方式在期望时间 $O(n\log n)$内可以构造平面上任意 n 个点的 Voronoi 图。

下面具体介绍 Voronoi 图中的点定位问题。修改上述算法使得在阶段 i 时保持的不是 $\mathrm{Vor}(S^i)$ 而是某种三角剖分 $H(S^i)$，这只要把点集 S 中的点与该点的 Voronoi 域的顶点连接起来便得到一种三角剖分 $H(S^i)$，称 $H(S^i)$ 中的域为三角形△(某些无界域不是三角形)，如图 10-10(d)所示。重要的是 $H(S^i)$ 具有而 $\mathrm{Vor}(S^i)$ 不具备的下述有界度性质：$H(S^i)$ 中的每个域△由 S^i 中至多 4 个点确定。所谓确定点意指它的 Voronoi 域或者与△邻接，或者包含△的点。另外要求有界度性质保持结构的深度是对数的。

结构 $H(i)$ 现在是三角剖分 $H(S^0),H(S^1),\cdots,H(S^i)$ 序列的结构，它的叶指向 $H(S^i)$ 中的三角形。

考虑第 $i+1$ 次加入点 $s=s_{i+1}$。已知如何将 $\mathrm{Vor}(S^i)$ 修改为 $\mathrm{Vor}(S^{i+1})$，也可以把 $H(S^i)$ 变为 $H(S^{i+1})$，以及 $H(i)$ 变为 $H(i+1)$。下面阐述后者：当删去时，标记对应于 $H(S^i)$ 中被删去三角形的 $H(i)$ 的叶。比如图 10-10(e)中，删去时，标记与阴影域相交的所有三角形，还要定位 $H(S^{i+1})$ 中新生三角形的结点。图 10-10(f)中，这些三角形是与阴影域相邻或者包含在该域中。这些三角形分成两种类型：新 $\mathrm{Vor}(s)$ 域内部和外部的三角形。与 $\mathrm{Vor}(s)$ 相邻的任意 Voronoi 域可以包含至多三个新生外部三角形。给定一个被删去的三角形 $\delta\in H(S^i)$ 及新生三角形 $\gamma\in H(S^{i+1})$，如果 $\gamma\cap\delta\neq\varnothing$，则 γ 是 δ 的子结点。如果 γ 是一个外部三角形，那么称 γ 是 δ 的一个外部子结点；否则，是内部子结点。比如，图 10-10(f)中，α，β 和 γ 是 δ 的外部子结点，而 $\bar{\gamma}$ 是 δ 的一个内部子结点。$H(S^i)$ 中已删去的三角形至多有三个外部子结点。另外，仅与指向 $H(i+1)$ 的新生叶的外部子结点的 $H(i)$ 指针的叶联系。

依据绕 s 的循环顺序，连接 $H(S^{i+1})$ 中新生内部三角形的所有叶，建立对应于该循环表的顺时针排列的搜索树(利用随机二叉树)。这种搜索树称为与 s 有关的辐射树。给定位于 $H(S^{i+1})$ 中域 $\mathrm{Vor}(s)$ 内的任意点 p，利用辐射树在对数时间内以高概率可以确定包含 p 的 $\mathrm{Vor}(s)$ 中的内部三角形。

设 p 是任意询问点，定位包含 p 的 $H(S^i)$ 中三角形的过程如下：从 $H(i)$ 的根开始，根对应于 $H(S^0)$ 中的唯一无穷域，然后下降到包含 p 的根的子结点。不断重复该过程，直至达到对应于包含 p 的 $H(S^i)$ 中三角形结构的叶。利用 $l(s_j)$ 可以执行下降操作：给定结构中一个结点，它对应于由点 $s_j(j\leqslant i)$ 删去的包含 p 的三角形 δ。如果 δ 的一个外部子结点包含 p，则由 δ 指向其外部子结点的指针可以定位 p；否则，利用与 s_j 关联的辐射树在对数时间内可以定位包含 p 的三角形的内部子结点。

上述点定位的耗费与搜索路径的长度成比例，比例系数在一个对数因子内。由于 $H(i)$ 的深度是 $\tilde{O}(\log i)$，所以利用 $H(i)$ 进行点定位的耗费是 $\tilde{O}(\log^2 i)$。另外，S^i 中距询问点 p 的最近邻近点是标记包含 p 的 Voronoi 域的点。因此，利用结构 $H(i)$ 在 $\tilde{O}(\log^2 i)$ 时间内可以回答最近邻近询问。

10.2.4 构形空间

上述增量算法有许多共同点，本小节将抽象其共性，目的是更广泛地应用它。利用构形空间可以完成这个工作。构形空间是构成随机增量算法的基础。

给定对象的一个抽象集合 S，对象随具体问题的差异而不同，比如，处理四边形分解

时对象是线段，而处理凸多胞形时它们是半空间。

S 上的构形 σ 是一偶对 $(D,L)=(D(\sigma),L(\sigma))$，其中 D 和 L 是 S 的不相交的子集。D 中对象称为与 σ 有关的引发物(triggers)，又称它们是确定 σ 的对象。L 中的对象叫做与 σ 有关的限制物(stopper)，又称与 σ 冲突的对象。定义度 $d(\sigma)$ 是 $D(\sigma)$ 的基数，级或者冲突规模 $l(\sigma)$ 定义为 $L(\sigma)$ 的基数。为简单起见，用 d 表示度，l 代表级。S 上的构形空间 $\Pi(S)$ 是 S 上的构形集，它具有有界度性质：$\Pi(S)$ 中每个构形的度是有界的。

$\Pi(S)$ 是一个多子集，这意味着 $\Pi(S)$ 中多个不同的构形可以有相同的引发物和限制物。所谓 $\Pi(S)$ 的规模是指 $\Pi(S)$ 中多子集元素的总数，其中具有相同引发物和限制物的不同构形将重复计数。如果具有相同引发物和限制物的不同构形不重复计数，那么计数结果称为 $\Pi(S)$ 的简化规模。用 $|\Pi(S)|$ 或者 $\Pi(S)$ 表示 $\Pi(S)$ 的规模，$\tilde{\Pi}(S)$ 表示简化规模。对于整数 $i(i\geqslant 0)$，定义 $\Pi^i(S)$ 是具有级 i 的 $\Pi(S)$ 中构形的集合，用 $\Pi^i(S)$ 表示它的规模。$\Pi^0(S)$ 中的构形被称是 S 上激活的，$\Pi^i(S)$ 中的构形被称是 S 上具有级 i 部分激活的。下面举两个例子。

例 10-1 四边形分解

给定平面上线段集合 S，对于 S 中的某个子集 S'，如果四边形 $\sigma\in$ 四边形分解 $H(S')$，则称 σ 在 S 上是可行的。在构造 $H(S)$ 的过程(一次增加一条线段)中可能出现的四边形都是可行的。

假设 σ 是一个可行的四边形，与 σ 的边界邻接的 S 中的线段集称为与 σ 有关的引发物集 $D(\sigma)$，而与 σ 相交的 $S\backslash D(\sigma)$ 中的线段集合称为与 σ 有关的冲突集 $L(\sigma)$。比如，图 10-11中，可行四边形 σ 的引发物集是 $\{s_1,s_3,s_5\}$，冲突集是 $\{s_2,s_4\}$。用这种方法，可以把抽象构形 $(D(\sigma),L(\sigma))$ 与每个可行四边形 σ 联系起来。显然，$D(\sigma)$ 的规模小于或者等于 4，这样，便得到 S 上所有可行四边形的构形空间 $\Pi(S)$，当然，这里要假设 S 中线段处在一般位置上。

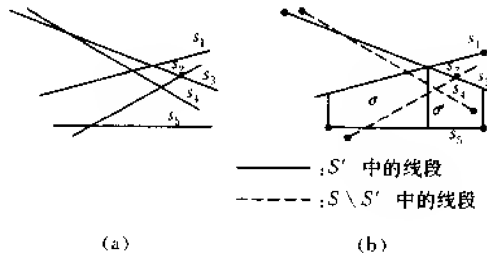


图 10-11

(a) 线段集 S (b) $H(S')$

在这个例子中，只保持未添加线段端点的冲突。在图 10-11(b)中，可行四边形 σ 和 σ' 与相同的引发物和限制物有关。还要注意，$\Pi^0(S)$ 中的构形恰好是四边形分解 $H(S)$ 中的四边形。

设 $S'\subseteq S$，如果构形 $\sigma\in\Pi(S)$ 并且 $D(\sigma)\subseteq S'$，则称 $\Pi(S')$ 是 $\Pi(S)$ 的子空间。另外，定义约束 $\sigma\downarrow S'$ 是 S' 上的构形，它的引发物集是 $D(\sigma)$ 并且冲突集是 $L(\sigma)\cap S'$，有

$$\Pi(S') = \{\sigma \downarrow S' \mid \sigma \in \Pi(S), D(\sigma) \subseteq S'\}$$

注意，$\Pi(S)$中两个不同的构形可以限制为$\Pi(S')$中具有相同引发物和限制物集的构形，但是两个受约束的构形必须是不同的。由于这个原因，视构形空间为多子集的是至关重要的。

所谓$\sigma \in \Pi(S)$在S'上是激活的，意指$\sigma \downarrow S'$属于$\Pi^0(S')$。σ在S'上是激活的当且仅当S'包含与σ有关的所有引发物但没有限制物。$\Pi(S')$中构形的级是指与S'有关的σ的冲突规模，比如，图 10-11 中，与S有关的可行四边形σ的冲突集是$\{s_2, s_4\}$。因此，与S有关的σ的冲突规模是 2。与S'有关的σ的冲突表是空的，所以与S'有关的σ的规模是零。这就是说，看作是$\Pi(S')$中构形的σ有零级。集合$\Pi^0(S')$由具有零级的$\Pi(S')$中的构形组成。这些对应于$H(S')$中的四边形，而这些四边形是由S'所形成的四边形分解中的四边形。此外，与S有关的$H(S')$中任意四边形σ的冲突规模是与σ相交的S/S'中线段的数目。

例 10-2 凸多胞形

设S是E^d中半空间集合，并且半空间处于一般位置上，$G(S)$是由S中半空间限界的超平面形成的排列，可以把抽象构形与$G(S)$的每个顶点σ联系起来。设引发物集$D(\sigma)$定义为S中半空间的集合，它的限界超平面包含σ，$D(\sigma)$的规模是d。冲突集$L(\sigma)$是S中半空间的集合，它的补包含σ。图 10-12(a)中，顶点q的引发物集是$\{s_1, s_4\}$，而限制物集是$\{s_6\}$。

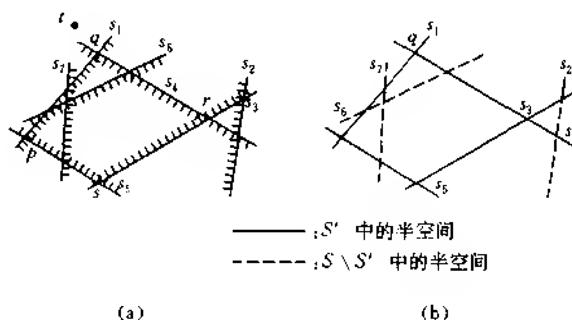


图 10-12 构形空间举例

(a) 半空间集 S (b) $S' = \{s_1, s_4, s_3, s_5\}$

用这种方法可以得到排列$G(S)$中顶点的构形空间$\Pi(S)$，并让$\Pi(S)$包含处于无穷远处的$G(S)$的顶点，这意味着在$G(S)$的无界边(1-面)的无穷远处的端点。图 10-12(a)中，t是这样的一个顶点。它的引发物集是$\{s_7\}$，而冲突集是$\{s_1, s_4, s_6\}$。

对于每个$S' \subseteq S$，构形空间$\Pi(S)$中的顶点σ在S'上是激活的，即它在$\Pi^0(S)$中当且仅当它是由交于S'中半空间所形成的凸多胞形的一个顶点。因此，$\Pi(S)$是由可能出现在$H(S)$的增量计算中的所有顶点组成的。

10.3 动态算法

10.2 节中介绍了四边形分解、Voronoi 图、凸多胞形等问题的静态和半动态(只允许增加对象)算法,本节将这些算法推广到动态环境,即允许增加对象、删去对象。我们的问题如下:设 M 表示任意时刻已存在的对象集合,目的是用动态方式保持几何划分 $H(M)$,而 $H(M)$ 依赖于所考虑的实际问题,在 M 中可以增加或者删去对象,要求能快速修改 $H(M)$ 及其有关的搜索结构。

考虑由增加和删去操作组成的修改序列 $\bar{u}$。定义它的特征 $\delta=\delta(\bar{u})$ 是+、-符号串,并且 $\delta(\bar{u})$ 中的第 i 个符号是+(或者-),当且仅当 $\bar{u}$ 中第 i 次修改是增加(或者删去)。设 S 是与 $\bar{u}$ 有关的对象集合,称 $\bar{u}$ 是一个 (S,δ) 序列。值得注意的是,并不是每个+、-符号序列 δ 都可以作为修改序列的特征。设 S 的规模是 n。

当 $\delta=+^n$ 时,修改序列 $\bar{u}$ 仅由增加操作组成。对于确定的 S 和 δ,如果 $\bar{u}$ 是从所有有效 (S,δ) 序列上依据均匀分布选取的,则称 $\bar{u}$ 是随机 (S,δ) 序列。如果 $\delta=+^n$,那么随机 $(S,+^n)$ 序列仅是通常的增加随机序列,我们用 $r(\delta,i)$ 表示在时间 i 当前(未删去的)对象的数目,当指定 δ 时,用 $r(i)$ 表示 $r(\delta,i)$。显然,$r(+^n,i)=i$。如果 $r(i)=\Omega(i)$,则称 δ 是弱单调的。设 $|\bar{u}|$ 表示 $\bar{u}$ 的长度,对于每个 $i\leqslant|\bar{u}|$,设 s_i 表示第 i 次修改中的对象,S^i 表示执行 $\bar{u}$ 期间在时间 i 未删去对象的集合。随机 (S,δ) 序列具有下述两个性质:如果 $\bar{u}$ 是随机 (S,δ) 序列,则有(1)对于任意 i、S 中每个对象等可能地是 s_i;(2)每个 S^i 是规模 $r(i)$ 的 S 的随机子集,其中 $r(i)\leqslant i$。

为简单起见,假设修改序列 $\bar{u}$ 中被删去的对象不能再添加。考虑随机 (S,δ) 序列的一种方法是:从左至右读 δ,如果符号是+,则从 S 中随机选取一个未添加的对象加入进去;如果符号是-,那么随机删去一个先前添加的对象。上述随机模型可以作为证明执行限界的一种方法,该执行限界适用于几乎所有的修改序列。这是因为在上述模型中 (S,δ) 序列上的概率分布是均匀的。称性质在大部分 (S,δ) 序列上成立,和称错误概率(即性质不成立的概率)与 n 的高阶多项式成反比例是相同的。一旦证明这一点,便可以假设所讨论的算法除极少数情况之外证实了那个执行限界。事实上,修改序列是否是随机的已无关重要,因为该模型仅用作证明上述性质的一个工具。我们希望证明以高概率成立的执行限界,目前仅在几种情况下是可能的,因此,这仍然是一个未解决的问题。

下面仅介绍将四边形分解的半动态算法推广到完全动态环境。设 S^k 表示时间 k 存在的未删去线段的集合,$H(S^k)$ 表示已得到的四边形分解,目的是在运算过程中保持 $H(S^k)$。另外,设 $s=s_{k+1}$ 是第 $k+1$ 次修改中所处理的线段,要求快速修改 $H(S^k)$ 为 $H(S^{k+1})$。

首先考虑第 $k+1$ 次修改是将 s 加入到 S^k 的情况。如果已知包含 s 的一个端点的 $H(S^k)$ 中的四边形 σ,那么利用 10.2 节中的算法将 s 加入到 $H(S^k)$。另外,在半动态环境下,借助于搜索结构 $H(i)$ 可以确定四边形 σ。搜索结构 $H(i)$ 亦适用于完全动态环境。

其次考虑第 $k+1$ 次修改是由 S^k 中删去 s 的情况,见图 10-13。首先将 s 表示为虚线,然后延伸与 s 相交的垂线至首次碰到 S^k 中线段,得到 $\hat{H}(S^k)$,见图 10-13(b)所示。最后删

去 s 及过 s 的端点，过 s 与 S' 中其他线段交点的垂直线段，便获得 $H(S^{k+1})$，如图 10-13(c)所示。这个修改过程可以在与 $\sum_f$ 面长(f)成比例的时间内完成，其中 f 是与 s 邻接的 $H(S^k)$中的所有四边形。

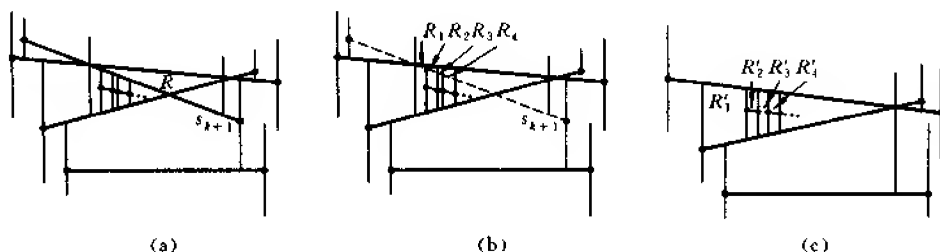


图 10-13 线段的删去

(a) $H(S^k)$ (b) $\hat{H}(S^k)$ (c) $H(S^{k+1})$

第 $k+1$ 次修改是加入还是删去完全由特征 δ 确定。如果是加入并且忽略定位 s 端点的耗费，那么其期望耗费与下式成比例

$$\frac{|H(S^{k+1})|}{|S^{k+1}|}=O\left(\frac{|S^{k+1}|+I_{k+1}}{|S^{k+1}|}\right)=O\left(1+\frac{I_{k+1}}{r(k+1)}\right) \tag{10-5}$$

其中 I_{k+1}是 S^{k+1}中线段之间交的数目。

其次，考虑第 $k+1$ 次修改是删去 s 的情况，该删去的期望耗费可以在与 $\sum_f$ 面长(f)成比例的时间内完成，即与下式成比例

$$\frac{|H(S^k)|}{|S^k|}=O\left(\frac{|S^k|+I_k}{|S^k|}\right)=O\left(1+\frac{I_k}{r(k)}\right) \tag{10-6}$$

其中 I_k 是 S^k 中线段之间交的数目，f 是与 s 邻接的 $H(S^k)$中的所有四边形。

引理 10-4 对于确定的 $j>0$，设 S^j 是规模为 j 的 S 的随机抽样，那么 S^j 中线段之间相交数 k_j 的期望值是 $O(kj^2/n^2)$，其中 k 是线段集中交的数目。

证明 考虑 S 中线段之间所有交的集合，记为 A。对于 A 中任意确定的交 v，定义一个 0-1 随机变量 I_v，使得 $I_v=1$，当且仅当 v 在 $H(S^j)$中。显然 $k_j=\sum I_v$，其中 v 在 S 中线段之间交的范围内。但是注意 v 在 $H(S^k)$中，当且仅当形成交 v 的两条线段在 S^j 中。发生这种情况的概率是 $O(j^2/n^2)$。因此，I_v 的期望值是 $O(j^2/n^2)$。引理结论成立。 证毕。

现估计随机(S,δ)序列上算法的期望总耗费。显然，对于每个 $k>0$，S^k 是规模为 $r(k)$ 的 S 的随机子集，由引理 10-4，S^k 中线段之间交的数目 I_k 的期望值是 $O(r(k)^2I/n^2)$，其中 I 是 S 中线段之间交的数目。因此，由式(10-5)和式(10-6)推得期望总耗费由下式限界

$$\sum_{k=1}^{|u|}\left(1+\frac{r(k)I}{n^2}\right)=n+\frac{I}{n^2}\sum_{i=1}^{|u|}r(k)=O(n+I) \tag{10-7}$$

这就是说，保持随机(S,δ)序列上四边形分解的期望耗费是 $n+\frac{I}{n^2}\sum_{i=1}^{|u|}r(k)=O(n+I)$，这不包含加入线段期间端点定位的耗费。

加入线段期间端点定位问题的解决也可以采用半动态环境中的搜索结构。设 $\bar{u}$ 是任

意(S,δ)序列，S^k表示时间k未删去线段的集合，搜索结构$H(k)$表示时间k的结构。$H(k)$的叶指向$H(S^k)$中的四边形。形式上，$H(0)$由对应于整个平面的一个结点组成。归纳假设已定义$H(k)$。设s_{k+1}表示第$k+1$次修改时被处理的线段，由$H(k)$构造$H(k+1)$的过程如下：

步 1 第$k+1$次修改期间$H(S^k)$中被删去的四边形设为σ，标记对应于σ的$H(k)$的叶为 killed。

步 2 定位$H(S^{k+1})$中新产生四边形的新结点。

步 3 建立$H(S^k)$和$H(S^{k+1})$之间连接的数据结构 link(s_{k+1})。

设$f\in H(S^k)$是被删去的四边形，而$g\in H(S^{k+1})$是新产生的四边形，如果$g\cap f\neq\varnothing$，则称g是f的一个子结点。

如果第$k+1$次修改是加入，$H(S^k)$中每个被删去的四边形至多有 4 个子结点，在这种情况下，link(s_{k+1})仅由$H(k)$的 killed 叶指向其子结点的指针组成。

如果第$k+1$次修改是删去，$H(S^k)$中被删去的四边形的子结点数不必由一个常数限界。如图 10-13 所示，被删去四边形$R\in H(S^k)$（图 10-13(a)）的子结点是$R_1',R_2',\cdots$，（图 10-13(c)），并且子结点数可以任意大。$H(S^k)$中任意被删去的四边形的子结点可以从左到右排列。设 link(s_{k+1})是由子结点的线性排序表组成，关于$H(k)$的每个被删去叶的表。借助于每个这样的线性排序表，保持一棵搜索树（即随机二叉树）。因此，在对数时间内可以搜索该线性顺序范围。设点p位于$H(S^k)$中被删去四边形内，利用 link(s_{k+1})在$O(\log|S^k|)=O(\log r(k))$时间内定位包含$p$的它的子结点。

利用$H(k)$定位点p的过程如下：从对应于整个平面的搜索结构$H(k)$的根开始，向下搜索到包含p的子结点并且继续重复该过程直至到达结构$H(k)$的一个叶为止。该叶对应于包含p的$H(S^k)$中的四边形。$H(k)$中从一个结点到它的子结点的下降可以利用与那个结点有关的连接结构来完成，这至多需要对数时间。因此推得定位$H(S^k)$中任意点的耗费与$H(k)$中搜索路径的长度成比例，比例因子为对数函数。

如果修改序列是随机的，那么搜索路径的期望长度可以和半动态环境中一样来估计。对于每个$j\leqslant k$，定义 0-1 随机变量v_j如下：如果包含点p的$H(S^{j-1})$中四边形在第j次修改期间被删去，定义v_j是 1；否则定义为 0。显然，搜索路径的长度是$\sum_{j\leqslant k}v_j$。如果第j次是删去，$v_j=1$当且仅当s_j是邻接于包含p的$H(S^{j-1})$中四边形的至多四条线段之一。S^{j-1}中线段很可能是等概率地被删去，因此删去每条线段的概率至多是$\frac{4}{|S^{j-1}|}=\frac{4}{r(j-1)}$。如果第$j$次修改是加入，$v_j=1$当且仅当$s_j$是邻接于包含$p$的$H(S^j)$中新产生的四边形的至多四条线段之一。$S^j$中线段很可能等概率地是$s_j$，因此加入线段的概率至多是$4/|S^j|=4/r(j)$。所以搜索路径的期望长度是$\sum_{j=1}^{|u|}\frac{1}{r(j)}$。对于大部分修改序列来说，搜索路径的期望长度是$O(\log k)$。

由于每次修改连接结构可以在与基本四边形分解中的结构变化成比例的时间内完成，所以保持搜索结构的耗费不可能使算法的总运行时间的增量超过一个常数因子。因此，式(10-7)的限界仍然成立。

在时间 k，点定位的期望耗费是 $O\left(\log k\sum_{j\le k}\frac{1}{r(j)}\right)$。对于大多数$(S,\delta)$序列来说，它是 $O(\log^2 k)$。

如果在 $\bar{u}$ 中某个对象已经被删去，则称 $H(k)$中这个对象为静止(dead)的。一般说来，在时间 k 静止对象的数目可能大大超过激活(未删去的)对象的数目。如果修改序列的特征 δ 不是弱单调的，那么基于算法的先前结构的执行不够令人满意。这是由于 $H(k)$可能依赖若干个过去静止的对象。为了克服这个缺陷，每当结构中静止对象的数目大大超过结构中激活对象的数目，比如说两倍时，我们就依据中间结果重建结构，通过按照先前添加它们的同样顺序添加当前的激活对象的方式完成重建。为此目的，使用半动态情况的算法，所得结构好像在先前修改期间从来没有遇到过静止对象。这之后，继续进行剩下的修改。每当需要时，使用这种方式不断重建该结构。经过重建的修改序列的特征 δ 将满足弱单调的要求，并且在任意给定时间静止对象的数目不可能超过激活对象数的两倍。

如果 $a(l)$表示第 l 次重建时激活对象的数目，那么 $\sum_l a(l)\leqslant|\bar{u}|$。这个结论是重建策略的实质。如果初始修改序列是随机的，而且任意重建操作中的添加也是随机的，那么单个重建的期望耗费可以同 10.2 小节半动态环境中一样估算。例如，估算四边形分解的动态保持的期望耗费如下：设 $I(l)$表示第 l 次重建期间被添加的 $a(l)$条线段之间交的数目。第 l 次重建的期望耗费依 10.2 节半动态环境估算是 $O(I(l)+a(l)\log a(l))$。因为初始修改序列是随机的，而且第 l 次重建中的线段集合是 S 的规模 $a(l)$的随机抽样，所以 $I(l)$的期望值是 $O(a(l)^2I/n^2)$，其中 I 是 S 中线段之间交的数目。又由于 $\sum_l a(l)\leqslant|\bar{u}|$，推得重建的总期望耗费是 $O(n\log n+I)$。再依据式(10-7)，有如下结论。

定理 10-6 保持随机(S,δ)序列上四边形分解的期望耗费是 $O(n\log n+I)$。

另外，需要定期重建删去结构中的静止对象。在任何时刻结构不包含静止对象。每当执行删去操作，就从结构中删去对象，一旦完成这一点，该结构看上去好像从来没有加入被删去的对象似的。在一般环境下，设 M 表示激活的，即在任意给定时刻未删去对象的集合，m 表示 M 的规模。概念上，新对象加入到结构 $H(M)$与半动态环境下的加入是一样的。利用旋转(rotation)操作可以从 $H(M)$中删去一个对象(这是随机二叉树情况的推广)。旋转的耗费依赖于旋转中的结构变化，由 M 随机删去期间平均结构变化近似于 $H(M)$的期望规模的$\frac{1}{m}$。

上面已给出四边形分解的一个动态点定位结构，利用旋转操作的删去技术可以建立四边形分解的另一个动态点定位结构 $H'(M)$，利用 $H'(M)$进行点定位，其耗费对于大多数修改序列是 $O(\log m)$。

使用结构 $H(M)$或者 $H'(M)$定位点的方法存在一个缺点，这就是该方法在大多数而不是所有修改序列上点定位的耗费是 $O(\log m)$。这意味着，如果 M 中线段的加入顺序是随机的，那么点定位的耗费在高概率的情况下是 $O(\log m)$。应用动态移动(dynamic shuffling)技术于四边形分解问题，即使用移动结构 $Sh(M)$，点定位的耗费对于每个修改序列是 $\tilde{O}(\log m)$。

由于篇幅有限，旋转操作，$H'(M)$与 $Sh(M)$等内容这里均不详述。

10.4 随机抽样

10.2 节中给出了求解几个问题的随机增量算法，本节将介绍随机分治算法。随机分治法的基本思想是：设 S 是实轴上点的集合，从 S 中随机选取一点 p，p 将实轴分成大致相等规模的两个子点集(域)。更一般地，设 R 是 S 的规模为 r 的随机抽样(子集)，这意味着，S 中每个点等概率地出现于 R 中。选取规模为 r 的随机抽样的一种方法如下：从 S 中随机选取一个点放入 R 中，然后独立地并随机地从剩下的 $n-1$ 个点中选择一个点作为 R 的第 2 个点，重复该过程直至从 S 中选取 r 个元素为止，称该过程是无放回的随机抽样。

用 $H(R)$ 表示由 R 形成的实轴的划分，定义 $H(R)$ 中任意区间 I 的冲突规模为位于 I 的 $S\backslash R$ 中点的数目。设 S 的规模是 n，则有下述结论。

定理 10-7 设 S 是实轴上 n 个点的集合，$R\subseteq S$ 是规模为 r 的随机抽样，则 $H(R)$ 中每个区间的冲突规模以概率大于 $\frac{1}{2}$ 为 $O([n/r]\log r)$。更一般地，固定任意 $c>2$，对于任意 $a\geqslant r>2$，$H(R)$ 中每个区间的冲突规模以概率 $1-O(1/a^{c-2})$ 小于 $\frac{cn\ln a}{(r-2)}$。换句话说，冲突规模超过 $\frac{cn\ln a}{(r-2)}$ 的概率是小的，即为 $O(1/a^{c-2})$。

证明略。

10.4.1 具有限界的构形空间

设 S 是平面上线段的集合，选取规模为 r 的随机抽样 $R\subseteq S$。用 $G(R)$ 表示由 R 形成的排列，定义排列 $G(R)$ 中任意 2-面的冲突规模(与 S 有关的)为与它相交的 $S\backslash R$ 中线段的数目，如图 10-14 所示。$G(R)$ 中任意 2-面的冲突规模以高概率为 $O([n/r]\log r)$ 吗？一般说来，答案是否定的。如果要求每个 2-面的边数有限，比如每个 2-面是四边形或者三角形，那么 $G(R)$ 中每个四边形的冲突规模的确是 $O([n/r]\log r)$，此时的 $G(R)$ 记为 $H(R)$。$G(R)$ 与 $H(R)$ 之间的区别是 $H(R)$ 具有有界度性质：$H(R)$ 中的每个四边形由若干线段确定。

如果共享同一个引发物集的构形空间 $\Pi(S)$ 中构形数由一个常数限界，则称 $\Pi(S)$ 已限界。这是对构形空间施加的一个约束。例如，设 S 是平面上线段的集合，$\Pi(S)$ 是 S 上可行四边形的构形空间，那么具有相同引发物集 T 的 $\Pi(S)$ 中所有可行四边形，可以用由 T 形成的四边形分解中的四边形来表示。由于 T 的规模是有界的，所以这样的四边形的数目也是有界的。

定理 10-8 设 $\Pi(S)$ 是有界的构形空间，n 是 S 的规模，d 是 $\Pi(S)$ 中构形的最大数(d 是有界的)。设 R 是 S 的规模为 r 的随机抽样，对于每个激活构形 $\sigma\in\Pi^0(R)$，与 S 有关的 σ 的冲突规模以概率大于 $\frac{1}{2}$ 至多是 $c(n/r)\log r$，其中 c 是一个足够大的常数。更一般地说，固定任意 $c>d$，对于任意 $a\geqslant r>d$，$\Pi^0(R)$(与 S 有关的)中每个构形的冲突规模以概率 $1-O(1/a^{c-d})$ 小于 $c(n\log a)/(r-d)$。换句话说，超过 $c(n\log a)/(r-d)$ 的冲突规模的概率

是 $O(1/a^{c-d})$。

证明略。

例 10-3 给定平面上的线段集合 S，将定理 10-8 应用到 S 上可行四边形的构形空间，推得对于任意随机抽样 $R \subseteq N$，四边形分解 $H(R)$ 中每个四边形以高概率具有 $O\left(\left[\frac{n}{r}\right]\log r\right)$ 冲突规模。

定理 10-8 中，假设随机抽样 R 由无放回的随机抽样来选择的。另外还有两种适用于定理 10-8 的随机抽样方法。

有放回的随机抽样：从 S 中随机地选取第 1 个元素，再从整个集合 S 而不是剩余的 $n-1$ 个元素集合中独立并随机地选取第 2 个元素，重复该过程 r 次。所得集合 R 的基数不一定是 r，因为一个元素可以被多次选取。这种随机抽样方法将产生微不足道的差异。

贝尔努利抽样：以成功概率 $p=\frac{r}{n}$ 取一枚硬币，计算机模拟该抽取如下：从集合 $\{1,2,\cdots,n\}$ 中选取一个随机数，如果这个数小于或者等于 r，则称抛硬币成功。对于 S 中每个元素，独立地翻动硬币。设 R 是抛硬币成功的元素集合，那么 R 的期望规模是 r。

定理 10-8 提供了一种随机划分的方法，从而导致几个搜索问题的分治算法。基于随机抽样的搜索结构可以用顶-向下的方式或者底-向上的方式建立，下面介绍这两种方式。

10.4.2 顶-向下的抽样

本小节介绍建立基于随机抽样的搜索结构的顶-向下的方法。顶-向下的最简单的搜索结构是随机二叉树，从给定点集 S 中选择一个随机点 p，它把 S 分成大致相等规模的两个子集 S_1 和 S_2，用 p 标记搜索树的根，该根的两个子树是由 S_1，S_2 递归建立的树。显然，这是一个分治过程。

S 计算几何中，随机二叉树已用于几个问题的求解(10.2 节)。为了使定理 10-8 成立，我们假设划分 $H(S)$ 具有有界度性质，即 $H(S)$ 的相关维的每个小面是由 S 中有限个对象确定的，其中 S 是几何对象的集合。

基于顶-向下抽样的搜索结构定义如下：

(1) 选择一个足够大常数规模 r 的随机子集 $R \subseteq S$。

(2) 建立 $H(R)$ 及 $H(R)$ 的搜索结构。

(3) 建立 $H(R)$ 的相关维的所有面△的冲突表，冲突的概念依赖于所考虑的问题。

(4) 对于每个 $\triangle \in H(R)$，递归地建立与△冲突的 S 中对象的集合 $S(\triangle)$ 的搜索结构。

(5) 建立用 ascent(S,R) 表示的提升结构，并按下述方式将它应用于询问中。

回答询问如下。初始询问是在集合 S 上。先在较小的集合 R 上回答询问，这可以利用与 $H(R)$ 有关的搜索结构来完成。设 $\triangle \in H(R)$ 是该较小询问的回答。然后，递归地回答冲突对象集合 $S(\triangle)$ 上的询问。最后，利用提升结构 $ascent(S,R)$ 设法确定集合 S 上的回答。下面给出与平面上线段排列有关的搜索问题的顶-向下抽样的应用。

设 S 是平面上 n 条线段的集合，$G(S)$ 表示由这些线段形成的排列。目的是建立一个点定位结构，使得给定一个询问点 p，在对数时间内确定包含 p 的 $G(S)$ 的面。我们将这个

问题转化为：假设另外给定平面上一个可能无界的三角形 Γ，目的是定位包含询问点 p 的交 $\Gamma \cap G(S)$ 中的面。

利用上面描述的顶-向下的递归方案定义数据结构。数据结构的根用 Γ 标记，根处存储整个排列 $G(S) \cap \Gamma$，并且在 $O(n^2)$ 时间和空间内可以用增量方法构造它。然后取规模为 r 的随机抽样 $R \subseteq S$，其中 r 是足够大的常数。构造 $G(R) \cap \Gamma$，并改进它，使得改进之后的划分具有有界度性质。这里可以利用四边形分解或三角剖分。设 $H(R)$ 表示 $G(R) \cap \Gamma$ 的典型三角剖分，用 $O(1)$ 时间可以构造它，因为 R 有常数规模。对于每个三角形 $\triangle \in H(R)$，用 $S(\triangle)$ 表示它的冲突表，也就是与 $\triangle$ 相交的 $S \backslash R$ 中线段的集合。因为 R 的规模是常数，所以在 $O(n)$ 时间内可以确定 $H(R)$ 中所有单纯形的冲突表：如果它们冲突的话，那么仅检验由 $H(R)$ 中的一个三角形和 $S \backslash R$ 中的一条线组成的每一个对。

由例 10-3 可以得到 $H(R)$ 中每个三角形的冲突规模以大于 $\frac{1}{2}$ 的概率小于 $b\left(\frac{n}{r}\right)\log r$，对于一个适当的常数 $b>1$，可以计算它。算法中检验随机抽样 R 的确满足这个性质。如果不满足，则取一个新的随机抽样 R，不断取新的抽样，直至该条件适用于每个三角形。一旦得到一个成功的抽样，便对每个三角形 $\triangle \in H(R)$ 重复。如果 $\triangle$ 的冲突规模小于适当选择的常数，那么就停止计算。否则，在与线段集合 $S(\triangle)$ 有关的 $\triangle$ 范围内重复计算。

点定位执行如下：设 p 是询问点，假设它位于与根级有关的三角形 Γ 中。为了定位 $G(S) \cap \Gamma$ 中的 p，先定位包含 p 的 $H(R)$ 中的三角形 $\triangle$。这需要常数时间，因为 R 的规模是不变的。然后递归地定位包含 p 的 $G(S(\triangle)) \cap \triangle$ 的面。与该面相关的父辈指针报告包含 p 的 $G(S) \cap \Gamma$ 的面。这是由于已把约束排列 $G(S(\triangle)) \cap \triangle$ 的每个面，与指向包含它存储在父辈级排列 $G(S) \cap \Gamma$ 中的面的父辈指针联系起来，这些父辈指针组成顶-向下抽样中的提升结构。

点定位的耗费满足下列递归关系：

$$q(n) = \begin{cases} O(1) & n \text{ 小于最低限界；} \\ O(1) + q\left(b\,\frac{n}{r}\log r\right) & \text{否则。} \end{cases}$$

如果选择抽样规模 r 是一个足够大的常数，则有 $q(n)=O(\log n)$。

给定集合 S，建立根为三角形 Γ 的搜索结构的期望时间耗费设为 $t(n)$，$t(n)=t(|S(\Gamma)|)$，$S(\Gamma)=S$，$t(n)$ 满足下列递归关系：

$$t(n) = \begin{cases} O(1) & n \text{ 小于最低限界；} \\ t(|S(\Gamma)|) & \text{否则。} \end{cases}$$

可以验证，$t(n)=O(n^2)+\sum\limits_{\triangle \in H(R)} t(|S(\triangle)|)=O\left(n^2+r^2 t\left(b\,\frac{n}{r}\log r\right)\right)$；由归纳法推得 $t(n) \leqslant n^2 c^{\log_r n}$，其中 $c \gg b$。直观上这是明显的，因为递归深度是 $O(\log_r n)$。对于实数 $\epsilon>0$，可以选择足够大的 r，使得 $t(n)=O(n^{2+\epsilon})$。设 $S(n)$ 表示点定位结构的规模，它满足 $t(n)$ 相同的递归关系，因此 $S(n)=O(n^{2+\epsilon})$。所以有下面的结论。

定理 10-9 给定平面上 n 条线段的任意排列及实数 $\epsilon>0$，可以在期望时间 $O(n^{2+\epsilon})$

内构造规模为$O(n^{2+\epsilon})$的点定位结构，并且询问时间为$O(\log n)$。

建立搜索结构的耗费和询问时间之间可以进行折衷，为了减小ϵ，必须使用一个较大的抽样规模。

10.4.3 底-向上的抽样

n条线段的排列规模是$O(n^2)$，该限界与定理10-9中的限界$O(n^{2+\epsilon})$之间有一间隙，通过改变抽样过程，即建立底-向上的搜索结构可以消去该间隙。基于底-向上抽样的最简单的搜索结构是跳越表(skip list)。

给定对象集S，目的是建立一个可以用于回答划分$H(S)$上询问的搜索结构。我们用Sample(S)表示与S有关的搜索结构，n表示S的规模。如果S是实轴上点的集合，则Sample(S)就是跳越表。

从集合S开始，得到集合序列

$$S = S_1 \supseteq S_2 \supseteq \cdots \supseteq S_{r-1} \supseteq S_r = \varnothing$$

其中S_{i+1}是由S_i用下述方法得到：对于S_i中每个对象独立地翻动一枚硬币并且仅保留抛硬币是成功的那些对象。上述集合的序列称为S的分层。与跳越表一样，该分层的长度是$\tilde{O}(\log n)$并且分层中集合的总规模是$\tilde{O}(n)$，这是因为每个S_i的规模大致是S_{i-1}的一半。Sample(S)的状态将由S的分层来确定。Sample(S)由r级组成，其中r是上述分层的长度。可以把集合$S_i(1\leqslant i\leqslant r)$视为存储在第$i$级的对象集合。在每级$i$，存储由$S_i$形成的划分$H(S_i)$。$H(S_i)$的每个面和与它冲突的$S_{i-1}\backslash S_i$中对象的冲突表有关。此外，相继级$i$和$i+1$之间保留某个下降结构，用descent$(i+1,i)$表示。

假设划分$H(S)$具有有界度性质，Sample(S)的一个重要性质如下：由于S_i是S_{i-1}大致一半的规模的随机抽样，所以每个$\triangle\in H(S_i)$的冲突规模是以高概率为对数的。

回答询问如下：假设要回答与对象$S=S_1$集合有关的询问。首先回答与S_r有关的询问。由于S_r是空的，这个回答是平凡的。该回答之后，在数据结构中下降。归纳假设已有与集合$S_i(1\leqslant i\leqslant r)$有关询问的回答，该回答是$H(S_i)$的一个面$\triangle$。利用下降结构descent$(i,i-1)$及与$\triangle$有关的冲突信息寻找关于集合$S_{i-1}$的回答。要求从第$i$级到第$i-1$级的下降耗费多重对数时间。用这种方法逐级下降，最后得到与集合$S=S_1$有关的所要求的回答。

下面介绍底-向上抽样在线段排列问题中的应用。

设S是平面上n条线段的集合，$G(S)$表示S的排列，$G(S)$的凸域不必有若干有界侧面。改进$G(S)$使之具有有界度性质。这里用四边形分解代替三角剖分可以节省搜索时间。设$H(S)$是$G(S)$的四边形分解，我们利用底-向上抽样将点定位结构与$H(S)$联系起来，并用Sample(S)表示该搜索结构。

直接应用底-向上抽样得到Sample(S)。我们用抛硬币的方法分层S，设$S=S_1\supseteq S_2\supseteq\cdots\supseteq S_{r-1}\supset S_r=\varnothing$是所得到的序列。在每级$l(1\leqslant l\leqslant r)$存储四边形分解$H(S_l)$，并存储与每个四边形$\triangle\in H(S_l)$相交的$S_{l-1}\backslash S_l$中线段的冲突表$L(\triangle)$。最后定义下降结构descent$(l+1,l)$为由彼此叠加$H(S_l)$和$H(S_{l-1})$所得到的划分$H(S_{l+1})\oplus H(S_l)$，如图10-14所示。直观上，这是通过中间划分$H(S_{l+1})\oplus H(S_l)$从$H(S_{l+1})$下降到$H(S_l)$。另

外，还要把 descent$(l+1,l)$中的每个四边形与包含它的 $H(S_l)$中唯一四边形的指针联系起来。

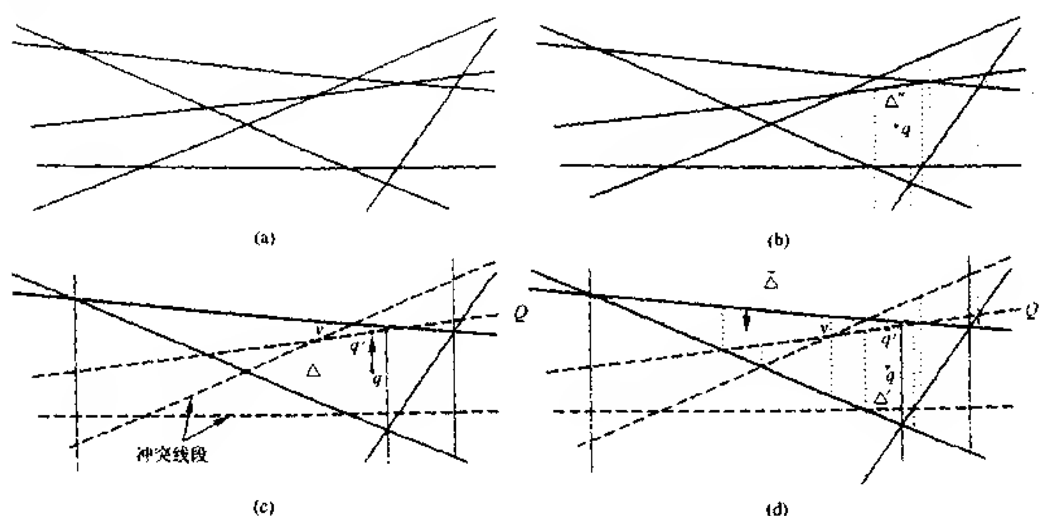


图 10-14　底-向上抽样在线段排列中的应用

(a) S_l　(b) $H(S_l)$　(c) $H(S_{l+1})$　(d) Descent$(l+1,l)$

Sample(S)用于点定位的过程如下：设 q 是询问点，$S_r=\varnothing$。定位 q 在 $H(S_r)$中是平凡的。归纳假设已定位 q 在 $H(S_{l+1})$中，$1\leqslant l\leqslant r$。我们希望从级 $l+1$ 下降到级 l。令$\triangle=\triangle_{l+1}$是包含询问点 q 的 $H(S_{l+1})$中的四边形，$\tilde{\triangle}$表示$\triangle$内叠加划分 descent$(l+1,l)$的约束。假设以可采用的$\tilde{\triangle}$方式存储 descent$(l+1,l)$。首先确定与自 q 向上垂直线相交的 S_l 中的第一条线 Q(设 q' 是交点)，如图 10-14(c)所示。显然，Q 或者是$\triangle$的上侧面的线，或者属于 $L(\triangle)$。因此，在 $O(|L(\triangle)|+1)$时间内可以平凡地确定 Q。令 v 是 Q 与 $L(\triangle)$或者$\triangle$内的一条线的交点并且离 q' 最近，在 $O(|L(\triangle)|+1)$时间内确定 v。假设对每个这样的 Q，有 Q 与 $L(\triangle)$中其他线的交的可行表。这只要扩大$\tilde{\triangle}$的表示。一旦确定了 v，在 descent$(l+1,l)$内由 v 移到 q'，如图 10-14(d)所示。这就定位了 q' 并且也随之确定了包含 q(图 10-14(d))的四边形$\triangle'\in$descent$(l+1,l)$。由 v 移到 q' 的耗费是 $O(|L(\triangle)|+1)$。包含 q 所要求的四边形$\triangle''=\triangle_l\in H(S_l)$是包含$\triangle'$的唯一四边形，如图 10-14(b)所示。

从级 $l+1$ 下降到级 l 耗费 $O(|L(\triangle_{l+1})|+1)$时间。由于 S_{l+1}是 S_l 大约一半规模的随机抽样，所以对于所有的 l，$|L(\triangle_l)|=\tilde{O}(\log n)$。另外，数据结构中的级数也是 $\tilde{O}(\log n)$，因此，定位点 q 所需要的时间是 $\tilde{O}(\log^2 n)$。但是对于一个固定的询问点 q，$\sum_{l\geqslant 1}|L(\triangle_l)|=\tilde{O}(\log n)$，其中$\triangle_l$ 是 $H(S_l)$中包含 q 的四边形。所以询问时间是 $\tilde{O}(\log n)$。可以证明下面的定理。

定理 10-10　建立 Sample(S)需要 $\tilde{O}(n^2)$时间和空间，定位点的耗费是 $\tilde{O}(\log n)$。

10.4.4　动态抽样

10.4.2 和 10.4.3 小节介绍了随机抽样如何用于几何搜索中，其搜索结构是静态的。

本小节将采用动态抽样技术使搜索结构成为动态的，也就是在搜索结构中允许增加或者删去一个对象。这里仅介绍基于底-向上抽样的搜索结构的动态化。

动态抽样的基本思想是用动态方式模拟静态抽样。设 M 表示在任意给定时间未删去对象的集合，m 是 M 的规模，其中对象依赖于所考虑的问题。构造搜索结构 Sample(M) 如下：首先产生一个分层 $M=M_1\supseteq M_2\supseteq\cdots\supseteq M_{r-1}\supset M_r=\varnothing$，其中由 M_i 得到 M_{i+1} 是通过对 M_i 中每个对象独立地翻转硬币并仅保留抛硬币获得成功的那些对象。Sample(M)的状态完全由 M 的上述分层来确定，它由 $r=\tilde{O}(\log m)$ 级组成，每级 l 与划分 $H(M_l)$ 有关，该划分依赖于所考虑的问题。descent$(l+1,l)$表示相继级 $l+1$ 与 l 之间的下降结构，询问回答期间要使用 descent$(l+1,l)$。

动态抽样技术的关键是 M 的分层，并因此，Sample(M)的状态将与实际上建立 Sample(M)的修改序列无关。这意味着在静态环境下所得到的关于询问时间的限界在动态环境下仍然成立。

加入新对象到 Sample(M)的过程如下：设 $M'=M\cup\{s\}$，要求由 Sample(M)得到 Sample(M')。重复抛一枚硬币直至失败（反面朝上），设 j 是失败前所得到的成功（正面朝上）数目，再通过 $j+1$ 把 s 加入到 1 级。这就是说，对于 $1\leqslant l\leqslant j+1$，设 M'_l 表示 $M_l\cup\{s\}$，加入 s 到第 l 级是指修改 $H(M_l)$ 为 $H(M'_l)$ 并且相应地修改 descent$(l+1,l)$，这要求 descent$(l+1,l)$也是动态的。当 s 已加入到所有 1 至 $j+1$ 级时便得到已修改的数据结构 Sample(M')，它完全决定于 M' 的分层 $M'_1\supseteq M'_2\supseteq\cdots$，其中 $M'_l=M_l, l>j+1$。注意，上述 M' 的分层与建立 Sample(M')的先前修改的实际序列仍然无关。

从 Sample(M)中删去对象 s 的过程如下：设 $M'=M\setminus\{s\}$，要求由 Sample(M)得到 Sample(M')，这只需从包含 s 的 Sample(M)的所有级中删去它。也就是说，对于某个 $j\geqslant 0$，假设 s 通过 $j+1$ 出现在 1 级中。对于 $1\leqslant l\leqslant j+1$，设 M'_l 表示 $M_l\setminus\{s\}$，由第 l 级删去 s，其含义是修改 $H(M_l)$ 为 $H(M'_l)$。同时还要修改 descent$(l+1,l)$。在该过程的末尾处得到 Sample(M')。Sample(M')的状态由分层 $M'=M'_1\supseteq M'_2\supseteq\cdots$ 确定，其中 $M'_l=M_l, l>j+1$。同样，M' 的上述分层与建立 Sample(M')的先前修改的实际序列无关。

下面介绍该动态技术应用于线段排列的先前定义的点定位结构。

从 10.4.3 小节中描述线段排列的搜索结构的动态化开始。设 M 表示的意义同上，$G(M)$表示由 M 形成的排列，$H(M)$表示 $G(M)$的四边形分解，Sample(M)表示与 M 有关的搜索结构。把 10.4.3 小节中的静态方法应用于集合 M 来构造 Sample(M)，它有 $r=\tilde{O}(\log m)$级。各级 $l(1\leqslant l\leqslant r)$都与四边形分解 $H(M_l)$有关。另外，还给每个四边形 $\triangle\in H(M_l)$一个与它相交的 $M_{l-1}\setminus M_l$ 中线段的冲突表 $L(\triangle)$。最后，下降结构 descent$(l+1,l)$是由彼此重叠 $H(M_l)$和 $H(M_{l-1})$所得到的划分 $H(M_{l+1})\oplus H(M_l)$，如图 10-14 所示。另外，还给予 descent$(l+1,l)$中每个四边形一个指向包含它的 $H(M_l)$中唯一四边形的指针。

新线段 s 加入 Sample(M)的过程如下：重复抛硬币直到失败，设 j 是失败前所得到的成功数目，目的是通过 $j+1$ 给 1 级加入 s。对于 $1\leqslant l\leqslant j+1$，设 M'_l 表示 $M_l\cup\{s\}$。给第 l 级加入 s 是指把 $H(M_l)$修改为 $H(M'_l)$并且相应地修改 descent$(l+1,l)$。线段 s 加入到

$H(M_l)$需要 $O(m_l)$时间，其中 m_l 是集合 M_l 的规模。下面仅需考虑 descent$(l+1,l)$，$1\leqslant l\leqslant j+1$。

首先考虑 $l=j+1$，此时把线段 s 加入到 M_l 而不是 M_{l+1}。设$\triangle\in H(M_{l+1})$是与 s 相交的任意四边形，$\widetilde{\triangle}$是关于$\triangle$的 descent$(l+1,l)$的约束。对于每个这样的四边形$\triangle$，需要修改$\widetilde{\triangle}$，其过程如下：首先，把 s 加入到四边形分解$\widetilde{\triangle}$，这意味着$\widetilde{\triangle}$中与 s 相交的垂直辅助线被缩短，比如图 10-15(a)中过交点 x 的辅助线被缩短，其结果见图 10-15(b)。通过 s 上新的交点增加垂直辅助线，见图 10-15(b)中的 u 和 w。把 s 加入到$\widetilde{\triangle}$的耗费是 $O(|L(\triangle)|+1)$。

其次考虑 $l<j+1$ 的情况，s 加入到 M_l 以及 M_{l+1}。这时仅需将 s 加入到$\widetilde{\triangle}$的上述过程扩展如下：首先，如果 s 与$\triangle$的下(或者上)边界相交，那么通过交点 w 的垂直辅助线必须延伸到$\triangle$的上(或者下)边界，如图 10-15(c)所示。这是因为 w 属于第 $l+1$ 级有关的新四边形分解，因此在 descent$(l+1,l)$中，过 w 的垂直辅助线切过不属于第 $l+1$ 级的 $L(\triangle)$中的线。过 w 添加这样一条延伸的垂直辅助线的耗费是 $O(|L(\triangle)|+1)$。最后，s 与$\triangle$的左或者右边界相交，那么该边界应适当缩短，如图 10-15(d)所示。这是因为该边界对应于穿过原分解 $H(M_{l+1})$中交点 v 的垂直辅助线，因此，通过将 s 加入第 $l+1$ 级可以使它分裂。

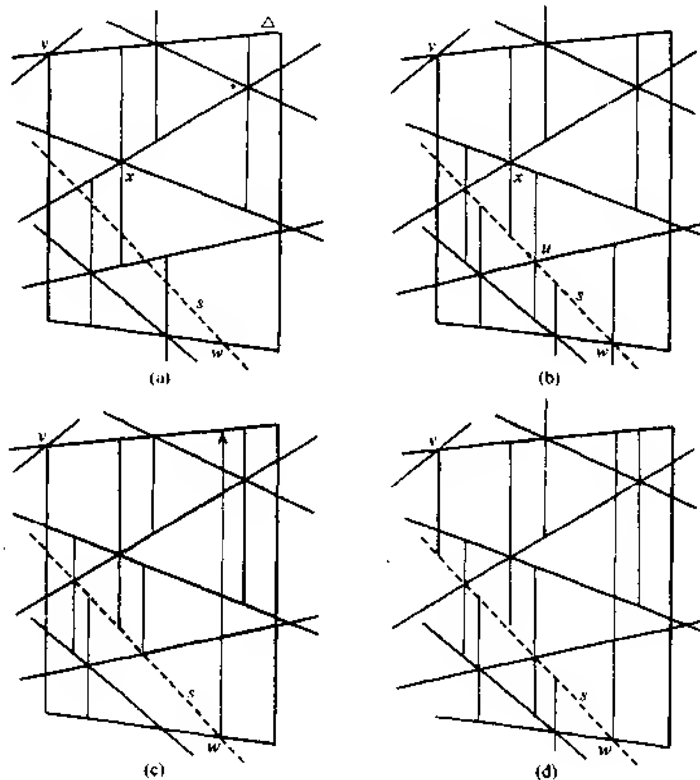


图 10-15　加入一条新线段 s

以加入过程的相反次序执行删去操作，这里不赘述了。

定理 10-11 设 $G(M)$ 是平面上 m 条线段的排列，存在允许 $\tilde{O}(\log m)$ 询问时间的 $\tilde{O}(m^2)$ 规模的动态点定位结构，该结构允许在 $\tilde{O}(m\log m)$ 时间内插入或者删去一条线段。修改的期望耗费是 $O(m)$。

10.5 并行几何算法

本节介绍并行几何算法，我们只涉及若干几何问题的并行算法设计及复杂性，而对另一些几何问题的并行算法只列出文献的线索，由于篇幅有限，这里不介绍并行算法的一般内容。

设计并行算法的目的是研究并行计算方法，要求该方法用尽可能少的处理机和尽可能少的计算时间求解给定的几何问题。众所周知，并行算法设计与并行计算机的类型密切相关，因此我们将以并行随机存取机(PRAM)作为机器模型。PRAM 是一个同步并行机模型，其中所有处理机都与一个共享随机存储器连接，并且它们之间的全部通信都要通过该共享随机存储器进行。PRAM 模型能够模拟并行的基本性质，此外，它也是一种被广泛接受的并行计算模型，并且其他合理的并行计算模型(比如超立方体模型，度限界网络模型等)可以模拟 PRAM。

假设任何数量的处理机可以同时读出存储器中同一信息(同一存储单元中的信息)，在执行写入操作时所有处理机同时向同一存储单元写入，这显然要产生写冲突，依据处理写冲突的不同方法可以把 PRAM 模型分成几种类型。第一种类型是 EREW(exclusive read，exclusive write) PRAM 模型，该模型不允许并发存取。第二种类型是 CREW (concurrent read，exclusive write) PRAM 模型，它要求在计算过程的任何一步中只有一台处理机写入一特定存储单元，但多台处理机可以同时从同一存储单元读出。另外还有 CRCW 模型，它允许并发读，又可以在同一时间内多台处理机写入同一存储单元：Common Write 模型，在同一时间内多台处理机写入同一存储单元是合法的当且仅当它们全部写入相同的值；Priority Write 模型，如果在同一时间内有多台处理机写入同一存储单元，那么标记最小的处理机写入成功。我们还可以定义 Rand EREW 模型，该模型除执行通常的算术运算和比较操作之外，每台处理机在计算过程的一步中可以产生 1 至 n 中的一个随机数。

如果并行算法中使用的处理机数目 $p(n)\leqslant f(n)$ ($f(n)$ 是输入规模 n 的多项式)，算法所需要的时间步数 $\leqslant f(\log n)$，即 $p(n)\in O(n^k)$，k 是常数，而最坏情况下的时间复杂性 $T(n)\in O((\log n)^m)$，m 是常数；则称该并行算法所需处理机数量-时间限界为多项式-对数时间。有时还以处理机数量限界与并行时间限界的乘积作为并行算法性能的度量标准，称为并行算法的工作限界。如果并行算法中某些步骤是由串行子算法完成的，那么还应考虑串行子算法的时间复杂性。

一般并行算法设计技术中的并行分治、构造与搜索方法等也适用于计算几何。下面举两个例子，其中的并行算法均为周培德提出。

例 10-4 并行分治。该技术划分问题成两个或多个子问题，以并行方式分别求解这

些子问题，然后合并子问题的解便得到原问题的解。如何划分问题成足够多的并行过程是非常困难的，并且不是所有问题都能做到这一点。作为该方法应用的一个例子，考虑构造平面上 n 个点的点集 S 的上凸壳问题，这些点依其 x 坐标被分类。划分表 S 或 $\sqrt{n}$ 个邻接子表，每个子表的规模是 $\sqrt{n}$，并且对每个子表分别构造该子表中点的上凸壳。给每对子表赋一台处理机，用于计算相应两个上凸壳的共同上切线，利用二叉搜索计算在 $O(\log n)$ 时间内可以完成这个工作。然后确定 S_i 的上凸壳上哪些顶点属于 S 的上凸壳，便完成 S 的上凸壳构造工作。具体步骤如下：

步 1 划分 S 为 $\sqrt{n}$ 个子问题 $S_1, S_2, \cdots, S_{\sqrt{n}}$，每个子问题的规模为 $\sqrt{n}$。

步 2 用 $\sqrt{n}$ 台处理机并行计算 $\sqrt{n}$ 个子问题的上凸壳，设为 $\mathrm{CH}'(S_1), \mathrm{CH}'(S_2), \cdots, \mathrm{CH}'(S_{\sqrt{n}})$。

步 3 用 $\sqrt{n}/2$ 台处理机并行计算 $\mathrm{CH}'(S_1)$ 与 $\mathrm{CH}'(S_2)$，…，$\mathrm{CH}'(S_{\sqrt{n}-1})$ 与 $\mathrm{CH}'(S_{\sqrt{n}})$ 的共同上切线，设切线为 $t_1, t_2, \cdots, t_{\sqrt{n}/2}$，切点为 $a_1, a_2, \cdots, a_{\sqrt{n}}$。

步 4 $\dfrac{\sqrt{n}}{2}-1$ 台处理机并行计算 $\mathrm{CH}'(S_2)$ 与 $\mathrm{CH}'(S_3)$，…，$\mathrm{CH}'(S_{\sqrt{n}-2})$ 与 $\mathrm{CH}'(S_{\sqrt{n}-1})$ 的共同上切线，设切线为 $t'_1, t'_2, \cdots, t'_{\frac{\sqrt{n}}{2}-1}$，切点为 $b_1, b_2, \cdots, b_{\sqrt{n}-2}$。

步 5 依据切点 a_2 与 b_1，a_3 与 b_2，…，$a_{\sqrt{n}-1}$ 与 $b_{\sqrt{n}-2}$ 的相对位置同时确定是否删去 S_i 及 $\mathrm{CH}'(S_i)$（$i \in 2^{\{2,3,\cdots,\sqrt{n}-1\}}$）。

步 6 如果删去 S_i 及 $\mathrm{CH}'(S_i)$（$i \in 2^{\{2,3,\cdots,\sqrt{n}-1\}}$），则将每对 $\mathrm{CH}(S_{i-1})$ 与 $\mathrm{CH}(S_{i+1})$ 分配给一台处理机，由该台处理机计算 $\mathrm{CH}(S_{i-1})$ 与 $\mathrm{CH}(S_{i+1})$ 的共同上切线及切点。重复执行步 5、步 6，直至没有 S_i 及 $\mathrm{CH}'(S_i)$ 被删去，连续的共同切线及未被删去的 $\mathrm{CH}(S_i)$ 上 b_{i-1} 与 a_i 之间的边构成 S 的上凸壳。

上述算法记为 $Z_{10\text{-}1}$ 算法。

在步 5 至步 6 的循环中，均用 a_i、b_i 代表新切点，实际上每轮循环之后切点都可能发生改变。重复步 5 与步 6 的次数为 $O(\log\sqrt{n})$，利用二叉搜索在 $O(\log\sqrt{n})$ 时间内计算切线，所以算法的并行时间复杂性为 $O(\log^2\sqrt{n})$，算法并行步所需处理机台数为 $\sqrt{n}=O(\sqrt{n})$，算法的工作限界为 $O(\sqrt{n}\log^2\sqrt{n})$。算法的串行时间复杂性为 $O(\sqrt{n}\log\sqrt{n})$。依 x 坐标并行排序时间为 $O(\log n)$，并行排序使用 n 台处理机。

例 10-5 构造与搜索。该技术是适用于并行计算几何的另一个重要方法，它是先利用平面扫描技术产生串行算法，然后改为并行算法的一个实例。构造与搜索方法求解一个问题的解分为两个阶段：一个是构造阶段，在这个阶段中，依据问题中的几何数据并行构造数据结构；另一个是搜索阶段，即并行搜索构造阶段建立的数据结构，求得给定问题的解。该方法的一个应用例子是四边形分解问题：给定平面上 n 条不相交线段的集合 S，确定从每条线段端点引出的垂线与 S 中首条线段的交，如图 10-8 所示。求解这个问题的并行算法是，首先基于输入线段集合 S 并行构造数据结构，然后并行搜索该数据结构，在 $O(\log n)$ 时间内回答垂线与 S 中哪条线段首先相交的询问，下述算法适于在 CREW 模型上运行，算法步骤如下：

步 1 依据 S 中线段左、右端点的 x 坐标(y 坐标)并行排序，设为 $L_x^{左}=(x_1,x_2,\cdots,x_n)$，以此顺序编号 S 中的线段端点及线段；$L_y^{左}=(y_1,y_2,\cdots,y_n)$。同样处理右端点。

步 2 并行建立二叉树。从左至右扫描平面，坐标 x_1 所对应的点 p_1 作为二叉树根结点的标记，点 p_1 的 y 坐标 $y_1(=y_1')$分割 $L_y^{左}$ 为两部分：L_y' 和 L_y''。如果 x_2,x_3 对应的点 p_2、p_3 的 y 坐标 $y_2(=y_4')$，$y_3(=y_7')$分别小于、大于 $y_1(=y_1')$，则将 p_2、p_3 分别作为根的左、右子结点的标记。往后每轮并行处理 $L_x^{左}$ 中 2^m 个元素，$m=2,3,\cdots$，每轮处理中并行比较 $L_x^{左}$ 中元素对应点的 y 坐标与二叉树中已标记叶上端点 y 坐标的大小，还要并行确定给哪些结点标记。直至 $L_x^{左}$ 为空。此树称为 $T_{左}$，见图 10-16(b)。同理，用从右至左扫描平面方法建立二叉树 $T_{右}$，见图 10-16(c)。

步 3 并行搜索。$2n$ 台处理机并行确定过 $2n$ 个端点的垂线与 S 中哪条线段首先相交，这只要分别搜索 $T_{左}$ 和 $T_{右}$。比如确定过左端点的垂线与 S 中哪条线段首先相交的过程如下：过根结点即 p_1 点的垂线与 S 中所有线段不相交，过根的左、右子结点所标记端点的垂线只可能与以 p_1 为左端点的线段相交，见图 10-16。过 $L_x^{左}$ 中其他值所对应的端点的垂线通过搜索二叉树 $T_{左}$ 和 $T_{右}$ 可以判定与 S 中哪条线段首先相交，例如二叉树 $T_{左}$ 中标记 p_4 的结点所代表的端点 p_4，过 p_4 的垂线向上只可能与 p_1 为左端点的线段相交；过 p_4 的垂线向下只可能与 p_2 为左端点的线段相交。

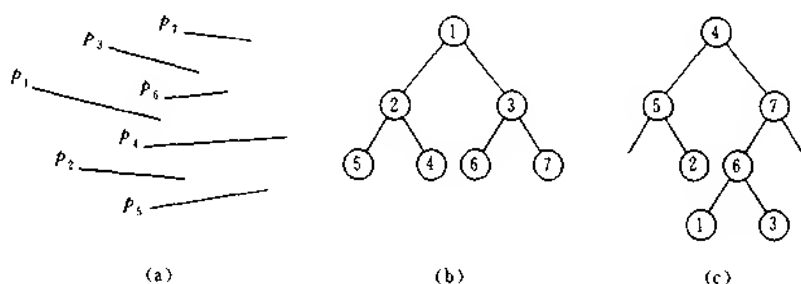


图 10-16 线段集及其端点的二叉树

(a) 不相交的线段集 (b) 二叉树 $T_{左}$ (c) 二叉树 $T_{右}$

上述算法记为 $Z_{10\text{-}2}$。

$Z_{10\text{-}2}$算法中步 1 的并行排序时间为 $O(\log n)$，用 $2n$ 台处理机；步 2 耗费 $O(\log n)$并行时间及 n 台处理机；步 3 耗费 $O(\log n)$并行时间及 $2n$ 台处理机。算法的工作复杂性为 $O(n\log n)$。

下面简单介绍 S 计算几何中几个典型问题的并行算法。

10.5.1 凸壳问题

给定 E^d 中 n 个点的集合 S，例 10-4 中的并行算法计算 $d=2$ 时的上凸壳，同理可以计算下凸壳，因此求得点集 S 的凸壳。另外，Miller 和 Stout(1988)提出的并行算法，将 S 分成 $n^{\frac{1}{4}}$个子问题，并行独立求解这些子问题，然后在 $O(\log n)$并行时间内合并所有子问题的解得到原问题的解。该算法所需时间满足下列递归关系式

$$T(n) \leqslant T(n^{1/4}) + O(\log n)$$

其解 $T(n)=O(\log n)$。

$d=2$ 时，还有许多其他并行算法求点集 S 的凸壳，现将这些并行算法列于表 10-1，该表中符号 $\tilde{O}(f(n))$ 表示以高概率成立的渐近复杂性限界。$d=3$ 时的各种并行算法也已列于表 10-1，这些算法也是利用分治策略设计的，只是三维中无预先分类的概念，所以划分步骤是非平凡的。Reif 和 Sen(1992)利用随机抽样执行划分以及 Amato、Goodrich 和 Ramos(1994)非随机化该方法的方法。而 Amato 和 Preparata(1993)使用并行分离平面的方法，后来这种方法被推广到高维。

表 10-1　并行凸壳算法

问　题	模　型	时　间	工　作	设计者及提出时间
二维预分类的	rand-CRCW	$\tilde{O}(\log^* n)$	$\tilde{O}(n)$	Ghouse 和 Goodrich(1991)
二维预分类的	CRCW	$O(\log\log n)$	$O(n)$	Berkman,Schieber 和 Vishkin(1996)
二维预分类的	EREW	$O(\log n)$	$O(n)$	Chen(1995)
二维多边形	EREW	$O(\log n)$	$O(n)$	Chen(1995)
二维	rand-CRCW	$\tilde{O}(\log n)$	$\tilde{O}(n\log h)$	Ghouse 和 Goodrich(1991)
二维	EREW	$O(\log n)$	$O(n\log n)$	Miller 和 Stout(1988)
二维	EREW	$O(\log^2 n)$	$O(n\log h)$	Ghouse 和 Goodrich(1991)
三维	rand-CRCW	$\tilde{O}(\log n)$	$\tilde{O}(n\log n)$	Reif 和 Sen(1992)
三维	CREW	$O(\log n)$	$O(n^{1+1/c})$	Amato 和 Preparata(1993)
三维	EREW	$O(\log^2 n)$	$O(n\log n)$	Amato,Goodrich 和 Ramos(1994)
三维	EREW	$O(\log^3 n)$	$O(n\log h)$	Amato,Goodrich 和 Ramos(1994)
固定的 $d\geqslant 4$	rand-EREW	$\tilde{O}(\log^2 n)$	$\tilde{O}(n^{\lfloor d/2\rfloor})$	Amato,Goodrich 和 Ramos(1994)
偶数的 $d\geqslant 4$	EREW	$O(\log^2 n)$	$O(n^{\lfloor d/2\rfloor})$	
奇数的 $d>4$	EREW	$O(\log^2 n)$	$O(n^{\lfloor d/2\rfloor}\log^c n)$	

与构造凸壳密切相关的问题是 d 维线性规划，对于这个问题也已设计了并行算法进行求解。当维数 d 固定时，用下述方法求解线性规划问题。首先将它转换成对偶问题，并构造相应对偶空间的凸壳，然后计算与该凸壳对偶的单纯形中每个顶点的值。但这种方法对于 $d\geqslant 4$ 将是无效的。求解线性规划问题的几种并行算法及其复杂性列于表 10-2。

表 10-2　d 维并行线性规划

模　型	时　间	工　作	设计者及提出时间
rand-CRCW	$\tilde{O}(1)$	$\tilde{O}(n)$	Alon 和 Megiddo(1990)
CRCW	$O((\log\log n)^{d+1})$	$O(n)$	Goodrich(1996)
EREW	$O(\log n(\log\log n)^{d-1})$	$O(n)$	Goodrich(1996)

10.5.2 排列与分解

排列与分解是另一类重要的几何问题，这类问题与划分空间的方式有关，现已有一些并行算法求解这类问题，见表 10-3。表中列出的问题及有关问题如下。

排列：由一批几何对象（比如线、线段或者高维中的超平面）的交所确定的空间划分。构造排列的算法产生由划分所确定的各种原始拓扑之间的所有邻接信息（比如交点、边、面等）的关联图。

红色-蓝色排列：给定几何对象集合 A 和 B，并且 $A(B)$ 中的对象不相交，由 A 与 B 的交所确定的空间划分。

轴平行：所有线段、线平行于一个坐标轴，由这些线段、线确定的空间划分。

多边形三角剖分：通过添加顶点之间的对角线把多边形内部分解成三角形。

四边形分解：见例 10-5，该例中描述的并行算法是求解四边形分解问题的并行算法。

星状多边形：从一个点完全可视的简单多边形，即具有非空核的多边形。

$\frac{1}{r}$ 切割：划分 E^d 成 $O(r^d)$ 个单纯形并且每个单纯形至多与 $\frac{n}{r}$ 个超平面相交。

表 10-3 并行排列与分解算法

问题	模型	时间	工作	设计者及提出时间
d 维超平面排列	EREW	$O(\log n)$	$O(n^d)$	Amato，Goodrich 和 Ramos(1994)
二维线段排列	rand-CRCW	$\tilde{O}(\log n)$	$\tilde{O}(n\log n+k)$	Clarkson，Cole 和 Tarjan(1992)
二维轴平行线段排列	CREW	$O(\log n)$	$O(n\log n+k)$	Goodrich(1991)
二维红色-蓝色线段排列	CREW	$O(\log n)$	$O(n\log n+k)$	Goodrich，Shauck 和 Guha(1992)
二维线段排列	EREW	$O(\log^2 n)$	$O(n\log n+k)$	Amato，Goodrich 和 Ramos(1995)
多边形三角剖分	CRCW	$O(\log n)$	$O(n)$	Goodrich(1992)
多边形三角剖分	CREW	$O(\log n)$	$O(n\log n)$	Goodrich(1989)
二维不交线段集四边形分解	CREW	$O(\log n)$	$O(n\log n)$	Atallah，Cole 和 Goodrich(1989)

表 10-3 中第二个算法是把随机算法与并行算法结合起来求解排列问题的一个典型例子。设 S 是平面上有 k 个线段对相交的线段集合，目的是构造由 S 产生的排列 $A(S)$。首先由一个随机抽样得到关于 k 的估计值 k'。其次选择取决于 k' 规模为 r 的随机子集 $R\subset S$。利用次最佳并行算法构造 $A(R)$，并对点定位以并行方式处理。然后利用并行点定位算法以及某些特殊技术找到与 $A(R)$ 的每个网格相交的线段。利用另一个次最佳并行算法计算与每个网格相交线段之间的可视信息。最后，并行合并所得到的网格。由于依据抽样使次最佳并行算法中的各种关键参数取小值，所以获得最优期望工作复杂性。

因为计算线段排列的所有算法的工作复杂性限界取决于输入规模和输出规模，所以这些算法是输出敏感的。在这种情况下，必须稍加扩充计算模型，使得如果需要的话便可

以增加处理机。然而在所有这些算法中，这种要求也许发生于单个主要的处理机，因此这种修改不是那种与我们的假设不同的修改(假设赋给问题的处理机数目可能是输入规模的一个函数)。当然，为了在现实并行计算机上求解一个问题，将利用串行方法模拟这些高效率的并行算法中的一个算法，从而获得最优加速。

与相交有关问题的分类是处理检测相交方法的问题分类。检测一批对象是否至少有一个交，常常比寻找所有的交更容易些。表 10-4 列出一些问题的并行交检测算法。

表 10-4 并行交检测算法

问　题	模　型	时　间	工　作	设计者及提出时间
2 个凸多边形	CREW	$O(1)$	$O(n^{1+\epsilon})$	Dadoun 和 Kirkpatrick(1989)
2 个星形多边形	CREW	$O(\log n)$	$O(n)$	Ghosh 和 Maheshwari(1991)
2 个凸多面体	CREW	$O(\log n)$	$O(n)$	Dadoun 和 Kirkpatrick(1989)

在 E^d 中给定 n 个超平面的集合，另一个重要的分解问题是构造 $\frac{1}{r}$ 切割，Goodrich 提出了运行时间为 $O(\log n \log r)$ 及工作限界 $O(nr^{d-1})$ 的 EREW 算法。

10.5.3 邻近

欧几里德空间是一个度量空间，并且欧几里德距离的概念在许多计算几何的应用中起重要作用，例如计算最近点对可以用于碰撞检测问题，正如计算点集 S 中每个点的最近邻近问题(称为所有最近邻近(ANN)问题)可以用于碰撞检测问题一样。也许与邻近概念有关的问题中最基本的问题是划分平面为域，并且由点集 S 中的点 p 定义每个域 $V(p)$，使得 $V(p)$ 中每个点距 p 比距 S 中其他点更近，这种划分是 Voronoi 图，它的对偶图是点集 S 的 Delaunay 三角剖分。对于 E^d 中点集 S、提升变换将 S 中的每个点 $(x_1, x_2, \cdots, x_d)$ 映射到点 $(x_1, x_2, \cdots, x_d, x_1^2+x_2^2+\cdots+x_d^2)$，构成 E^{d+1} 中的点集 S'。我们知道构造 $d+1$ 维空间中点集 S 凸多胞形的算法可以产生构造 d 维空间中 Voronoi 图的算法，这就是说，任意 $d+1$ 维凸壳算法意指 d 维 Voronoi 图算法，表 10-5 中列出的结果是用这种方法得到的构造 Voronoi 图的并行算法及求解其他邻近问题的并行算法。

表 10-5 并行邻近算法

问　题	模　型	时　间	工　作	设计者及提出时间
凸状况中二维 ANN	EREW	$O(\log n)$	$O(n)$	Cole 和 Goodrich(1992)
二维 ANN	CREW	$O(\log n)$	$O(n\log n)$	Callahan(1993)
二维 Voronoi 图	rand-CRCW	$\tilde{O}(\log n)$	$\tilde{O}(n\log n)$	Reif 和 Sen(1992)
二维 Voronoi 图	EREW	$O(\log^2 n)$	$O(n\log n)$	Amato、Goodrich 和 Ramos(1994)
线段的二维 Voronoi 图	CREW	$O(\log^2 n)$	$O(n\log^2 n)$	Goodrich、O'Dunlaing 和 Yap(1993)

表 10-5 中有两处需要进一步说明，一处是凸状况意指点集中的点都在其凸壳的边界上，另一处是线段的 Voronoi 图，即指由互不相交的线段集定义的 Voronoi 图，并且由点

p 到线段 s 的距离定义为由 p 到 s 上的一个最近点的距离。

10.5.4 几何搜索

给定几何对象集合 S，比如线段集合，由 S 划分空间成若干个域。考虑点定位问题，即要建立一种数据结构以便快速回答下述问题：给定点 p，要求报告从 p 引出的垂直射线碰到的 S 中的第一个对象。表 10-6 列出平面点定位的并行算法，询问时间均为 $O(\log n)$。

表 10-6 并行几何搜索算法

询问问题	模 型	时 间	工 作	设计者及提出时间
任意划分中的点定位	CREW	$O(\log n)$	$O(n\log n)$	Atallah，Cole 和 Goodrich(1989)
单调划分中的点定位	EREW	$O(\log n)$	$O(n)$	Tamassia 和 Vitter(1991)
三角划分中的点定位	CREW	$O(\log n)$	$O(n)$	Cole 和 Zajicek(1990)
d 维超平面排列中的点定位	EREW	$O(\log n)$	$O(n^d)$	Amato，Goodrich 和 Ramos (1994)
三角剖分多边形中的最短路径	CREW	$O(\log n)$	$O(n)$	Goodrich，Shauck 和 Guha (1992)
三角剖分多边形中的特定射线	CREW	$O(\log n)$	$O(n)$	Hershberger 和 Suri(1993)
线和凸多面体相交	CREW	$O(\log n)$	$O(n)$	Cole 和 Zajicek(1990)

表 10-6 中出现的术语解释如下：

任意平面划分：由仅在其端点相交的线段集确定的平面划分。

单调划分：单调划分是一种平面的连通划分，在该划分中每个面与垂线相交成一条线段而不是多条线段。

三角剖分划分：平面划分成三角形的连通划分，在该划分中角是划分的顶点。

多边形中的最短路径：多边形内两点之间的位于多边形内部的最短路径。

特定射线询问：询问从一特殊点沿特定方向引出的射线所碰到的第一个对象。

10.5.5 可视性和最优化

各种可视问题的并行算法列于表 10-7，其中 m 表示可视图中的边数。该表中第二个算法是计算点可视多边形的并行算法，这个算法是许多其他算法的子算法，并且它需要比相对简单的最佳串行算法更复杂的处理和分析。该并行算法是递归的，划分边界为 $n^{\frac{1}{4}}$ 条子链，并且从可视 x 的原点计算可视链。每条这样的链相对于 x 是星状的，即有效单调的。然而由此单调性不能在得到最优限界的合并步中尽快地贯穿可视链。相反，必须依据链是简单多边形边界的子链的事实来获得两条链的交的对数时间计算，这导致表 10-7 中 $O(\log n)$ 的时间复杂性。

表 10-8 中列出某些几何最优化问题的并行算法。现解释表中出现的术语：

面积最大的空矩形：给定平面上 n 个点的集合 S，点集 S 内面积最大的矩形 R，并且 R 内不包含 S 的点。

表 10-7 简单多边形的并行可视算法

问题	模型	时间	工作	设计者及提出时间
核	EREW	$O(\log n)$	$O(n)$	Chen(1995)
从一点可视	EREW	$O(\log n)$	$O(n)$	Atallah,Chen 和 Wagener(1991)
从一条边可视	CRCW	$O(\log n)$	$O(n)$	Hershberger(1992)
可视图	CREW	$O(\log n)$	$O(n\log^2 n+m)$	Goodrich,Shauck 和 Guha(1992)

表 10-8 并行几何优化算法

问题	模型	时间	工作	设计者及提出时间
面积最大的空矩形	CREW	$O(\log^2 n)$	$O(n\log^3 n)$	Aggarwal,Kravets,Park 和 Sen(1990)
多边形之间的最近可视对	CREW	$O(\log n)$	$O(n\log n)$	Hsu,Chang 和 Lee(1992)
内接、外切三角形面积最优	CRCW	$O(\log\log n)$	$O(n)$	Chandran 和 Mount(1992)
内接、外切三角形面积最优	CREW	$O(\log n)$	$O(n)$	Chandran 和 Mount(1992)

多边形之间的最近可视对:平面上两个不相交简单多边形之间的最近可视顶点对。

内接、外切三角形面积最优:给定凸多边形 P,确定 P 中内接最大面积的三角形及外切 P 的最小面积三角形。

算 法 索 引

（右侧数字为算法所在的节号）

参 考 文 献

1. 周培德．算法设计与分析．北京：机械工业出版社，1992
2. 周培德．计算中的基本理论与方法．北京：北京理工大学出版社，1997
3. 周培德．串匹配的一种算法．计算机研究与发展，1990，27(2)：35～37
4. 周培德．化网络系统为不交型的并行计算．数学的实践与认识，1990，(3)：32～37
5. 周培德．求凸壳顶点的一种算法．北京理工大学学报．1993，13(1)：69～72
6. 周培德．求解货郎担问题的几何算法．北京理工大学学报，1995，15(1)：97～99
7. 周培德．确定任意多边形凸凹顶点的算法．软件学报，1995，6(5)：276～279
8. 周培德．货郎担问题的几何解法．软件学报，1995，6(7)：420～424
9. 周培德．几何算法求解货郎担问题．计算机研究与发展，1995，32(10)：63～65
10. 周培德．确定任意多边形的核的算法．工程图学学报，1995，(2)：28～30
11. 周培德．判定点是否在多边形内部的算法．北京理工大学学报，1995，15(4)：437～440
12. 周培德．关于某些几何覆盖问题的算法．北京理工大学学报，1995，15(5)：21～25
13. 周培德．多边形内点集的三角剖分算法．北京理工大学学报，1995，15(5)：26～28
14. 周培德．任意多边形三角剖分的算法．北京理工大学学报，1995，15(5)：83～86
15. 周培德．平面点集三角剖分的算法．计算机辅助设计与图形学学报，1996，8(4)：259～264
16. 周培德，周忠平．求凸多边形直径的算法．工程图学学报，1996，(2)：29～32
17. 周培德．连接两个多边形成一条回路的算法．计算机研究与发展，1996，33(11)：865～868
18. 周培德．二维线性规划问题的非数值算法．北京理工大学学报，1996，16(6)：665～670
19. 周培德．判定点集是否在多边形内部的算法．计算机研究与发展，1997，34(9)：672～674
20. Zhou Peide. An algorithm for partitioning polygons into convex parts. Journal Beijing Institute of Technology, 1997, 6(4): 363～368
21. 周培德，王文明．确定两个任意多边形的并的算法．北京理工大学学报，1998，18(1)：87～91
22. 周培德，张欢．确定平面直线图完全单调链集的算法．长沙：计算机工程与科学，1999，21(1)：1～3
23. Agarwal P K, Edelsbrunner H, Schwarzkopf O, Welzl E. Euclidean minimum spanning trees and bichromatic closest pairs. Discrete and Computational Geometry, 1991, 407～422
24. Aggarwal A, Kravets D, Park J K, Sen S. Parallel searching in generalized Monge arrays with applications. In: Proc. 2nd Annu. ACM Symp. Parallel Algorithms Architect., 1990, 259～268
25. Aho AV, Garey M R, Hwang F K. Rectilinear Steiner trees: Efficient special-case algorithms. Networks 7, 1977, 37～58
26. Akl S G, Toussaint G T. Efficient convex hull algorithms for pattern recognition applications. Proc. 4th Int'l Joint Conf. on Pattern Recognition, Kyoto, Japan. 1978, 483～487
27. Alon N, Megiddo N. Parallel linear programming in fixed dimension almost surely in constant time. In: Proc. 31st Annual IEEE Symp. Found. Comput. Sci., 1990, 574～582
28. Amato N M, Goodrich M T, Ramos E A. Parallel algorithms for higher-dimensional convex hulls. In: Proc. 35th Annual IEEE Symp. Found. Comput. Sci., 1994, 683～694
29. Amato N M, Goodrich M T, Ramos E A. Computing faces in segment and simplex arrangements.

In: Proc. 27th Annual ACM Symp. Theory Comput., 1995,672～682

30. Amato N M, Preparata F P. An NC parallel 3D convex hull algorithm. In: Proc. 19th Annual ACM Symp. Comput. Geom., 1993,289～297
31. Andrew A M. Another efficient algorithm for convex hulls in two dimensions. Info. Proc. Lett., 1979(9),216～219
32. Armillotta A, Mummolo G. A heuristic for the Steiner tree problem with obstacles. Technical Report, Department of Industrial Design and Production, University of Bari, Italy, 1988
33. Asano Ta. Asano Te, Imai H. Shortest path between two simple polygons. Information Processing Letters, 1987(24),285～288
34. Asano Te. Asano Ta, Guibas L G, Hershberger J, Imai H. Visibility of disjoint polygons. Algorithmica 1. 1986. 49～63
35. Atallah M. Some dynamic computational geometry problems. Computers and Mathematics with Applications, 1985, 11:1171～1181
36. Atallah M J, Cole R, Goodrich M T. Cascading divide-and -conquer: A technique for designing parallel algorithms. SIAM J. Comput, 1989, 18:499～532
37. Atallah M J, Chen D Z, Wagener H. Optimal parallel algorithm for visibility of a simple polygon from a point. J. Assoc. Comput. Mach., 1991, 38:516～553
38. Aurenhammer F. Voronoi diagrams——a survey of a fundamental geometric data structure. ACM Comput. Survey, 1991, 23:345～405
39. Baker B. Shortest path with unit clearance among polygonal obstacles. Presented at the SIAM Conference on Geometrical Modelling and Robotics, 1985
40. Baltsan A, Sharir M. On shortest paths between two convex polyhedra. Technical Report 180, Computer Science Dept., Courant Institute, New York, 1985
41. Beasley J E. A greedy heuristic for the Euclidean and rectilinear Steiner problem . EJOR 58, 1992, 284～292
42. Beasley J E, Goffinet F. A Delaunay triangulation-based heuristic for the Euclidean Steiner problem. Networks 1994, 24:215～224
43. Ben-Or M. Lower bounds for algebraic computation trees. Proc. 15th ACM Annual Symp. on Theory of Comput., May 1983, 80～86
44. Bentley J L, Maurer H A. Efficient worst-case data structures for range searching. Acta Informatica, 1980, 13:155～168
45. Bentley J L, Ottmann T A. Algorithms for reporting and counting geometric intersections. IEEE Trans. on Computers, 1979, 28:643～647
46. Bentley J L, Shamos M I. A problem in multivariate statistic: Algorithms, data structure, and applications. Proceedings of the 15th Annual Allerton Conference on Communication, Control, and Computing, 1977, 193～201
47. Bentley J L, Shamos M I. Divide and conquer for linear expected time. Info. Proc. Lett., 1978, 7: 87～91
48. Bentley J L. Multidimensional divide and conquer. Comm. ACM, 1980, 23:214～229
49. Bentley J L. Experiments on geometric traveling salesman heuristics. ORSA J. Comput., 1992, 4 (4):387～411
50. Bentley J L, Friedman J H. Fast algorithms for constructing spanning trees in coordinate spaces.

IEEE Transactions on Computers, 1978, c-27: 97～105

51. Bentley J L, Shamos M. Divide and conquer in multidimensional space. Proc. of the 8th ACM Ann. Symp. on the Theory of Computation, 1976, 220～230
52. Bern M W. Network design problems: Steiner trees and spanning k-trees. Ph. D. Thesis, Computer Science Dept., Univ. of California at Berkeley, 1987
53. Bern M W, Carvaho M de. A greedy heuristic for the rectilinear Steiner problem. Technical Report, Computer Science Division, Univ. of California at Berkeley, 1985
54. Brondsted A. An introduction to convex polytopes, Springer-Verlag, Berlin, 1983
55. Brost R. Analysis and planning of planar manipulation tasks. Ph. D. Thesis, Carnegie-Mellon University. CMU-CS-91-149, 1991
56. Callahan P B. Optimal parallel all-nearest-neighbors using the well-seated pair decomposition. In: Proc. 34th Annual IEEE Symp. Found. Comput. Sci., 1993, 332～340
57. Canny J. A new algebraic methods for robot motion planning and real geometry. Proc. of the 28th IEEE Symp. on the Foundations of Computer Science, 1987, 39～48
58. Canny J, Reif J. New lower bound techniques for robot motion planning problems. Proc. of the 28th IEEE Symp. on the Foundations of Computer Science, 1987, 49～60
59. Cavendish J C. Automatic triangulation of arbitrary planar domains for the finite element method. Int'l J. Numerical Method in Engineering, 1974, 8: 679～696
60. Chand D R, Kapur S S. An algorithm for convex polytopes, JACM, 1970, 17(1): 78～86
61. Chang R C. and Lee R C T. An $O(n\log n)$ minimal spanning tree algorithm for n points in the plane. BIT, 1986, 26: 7～16
62. Chang S K. The generation of minimal trees with a Steiner topology, J. ACM, 1972, 19: 699～711
63. Chazelle B. Filtering search: A new approach to query-answering. SIAM J. Comput., 1986, 15: 703～724
64. Chazelle B. Lower bounds on the complexity of polytope range searching. J. Amer. Math. Sci., 1989, 2: 637～666
65. Chazelle B. Lower bounds for orthogonal range searching I. The Reporting Case, J. ACM, 1990, 37: 200～212
66. Chazelle B. Computational geometry: a retrospective. In: Du D Z, Hwang F(ed). Computing in Euclidean geometry, Singapore: World Scientific Publishing Co. Pte. Ltd., 1995, 4: 22～46
67. Chazelle B. Triangulating a simple polygon in linear time. Disc. Comput. Geom., 1991, 6: 485～524
68. Chazelle B. Computational geometry and convexity. Report CMU-CS-80-150, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA. Ph. D. Thesis from Yale University, 1980
69. Chazelle B, Edelsbrunner H. An optimal algorithm for intersecting line segments in the plane, J. ACM, 1992, 39: 1～54
70. Chazelle B M. How to search in history. Information and Control, 1985, 64: 77～99
71. Chazelle B M, Dobkin D P. Detection is easier than computation. Proc. 12th ACM Annual Symp. on Theory of Comput., May, 1980, 146～153
72. Chazelle B M, Gruibas L J. Fractional cascading: I A data structuring technique, Algorithmica, 1986, 1: 133～162
73. Chazelle B M, Incerpi J. Triangulating a polygon by divide-and -conquer. Proc. 21st Allerton

Conference on Comm. Control and Comput., Oct. 1983, 447～456 .

74. Chen D. Efficient geometric algorithms on the EREW PRAM. IEEE Trans. Parallel Distrib. Syst., 1995, 6: 41～47

75. Chew L P. Planning the shortest path for a disk in $O(n^2\log n)$time. Proc. of the 1st Ann. Symp. on Computational Geometry, 1985, 214～220

76. Chiang Y, Tamassia R. Optimal shortest path and minimum-link path queries between two convex polygons in the presence of obstacles. Presented at the Computational Geometry Workshop, Raleigh North Carolina, Oct. 1993

77. Clarkson K L. Faster expected-time and approximation algorithms for the geometric minimum spanning trees. Proc. of the 16th Ann. ACM Symp. on the Theory of Cmputing, 1984, 342～348

78. Clarkson K L. Approximation algorithms for shortest path motion planning. Proc. of the 19th Ann. ACM Symp. on the Theory of Computing, 1987, 56～65

79. Clarkson K L. Randomized geometric algorithms. In: Du D Z. Hwang F K(ed). Computing in Euclidean Geometry, volume 1 of Lecture Notes Series on Computing, Singapore: World Scientific, 1992, 117～162

80. Clarkson K L. Cole R, Tarjan R E. Randomized parallel algorithms for trapezoidal diagrams. Internat. J. Comput. Geom. Appl., 1992, (2): 117～133

81. Clarkson K L, Tarjan R E, Van Wyk C J. A fast Las Vegas algorithm for triangulating a simple polygon. Disc. Comput. Geom., 1989, (4): 423～432

82. Cockayne E J. On the efficiency of the algorithm for Steiner minimal trees. SIAM J. Appl. Math., 1970, (18): 150～159

83. Cole R, Goodrich M T. Optimal parallel algorithms for polygon and point-set problems. Algorithmica, 1992, (7): 3～23

84. Cole R, Zajicek O. An optimal parallel algorithm for building a data structure for planar point location. J. Parallel Distrib Comput., 1990, (8): 280～285

85. Dadoun N, Kirkpatrick D G. Cooperative subdivision search algorithms with applications. In: Proc. 27th Allerton Conf. Commun. Control Comput., 1989, 538～547

86. Dasarthy B, White L J. On some maximin location and classifier problems. unpublished lecture, Computer Science Conference, Washington D.C., 1975

87. Dasarthy B, White L J. A maximin location problem. Operations Research, 1980, (28): 1385～1401

88. Dijkstra E W. A note on two problems in connection with graphs. Numerical Mathematics, 1959, (1): 269～271

89. Dobkin D, Edelsbrunner H. Space searching for intersecting objects. Proc. 25th IEEE FOCS, 1984, 387～392

90. Dobkin D P, Lipton R J. Multidimensional searching problems. SIAM J. Comput., 1976, (5): 181～186

91. Dolan J, Weiss R, MacGregor Smith J. Minimal length tree networks on the unit sphere. Annals of Operations Research, 1991, (33): 503～535

92. Dorward S E. A survey of object-space hidden surface removal. Internat. J. Comput. Geom. Appl., 1991

93. Du D Z, Hwang F K. Steiner minimal trees for bar waves. Chinese J. of Mathematics, 1987

94. Du D Z, Hwang F K. The Steiner ratio conjectire of Gilbert and Pollak is true. Proc. Nat. Acad. Sci., 1990(87):9464~9466

95. Du D Z, Hwang F K, Weng J F. Steiner minimal trees for regular polygons. Discrete and Computational Geometry, 1987,(2):65~87

96. Duda R O, Hart P E. Pattern classification and scene analysis. Wiley-Interscience, New York, 1973

97. Dyer M E. Linear time algorithms for two- and three-variable linear programs. SIAM J. Comp., Feb, 1984,13(1):31~45

98. Edahiro M, Tanaka K, Hoshino T, Asano T. A bucketing algorithm for the orthogonal segment intersection search problems and its practical efficiency. Proc. 3rd ACM Comput. Geom., 1987, 258~267

99. Edelsbrunner H. Algorithms in combinatorial geometry, Berlin: Springer-Verlag, 1987

100. Edelsbrunner H. Intersection problems in computational geometry. Ph. D. Thesis, Rep. 93, IIG, Technische Universitat Graz, Austria,1982

101. Edelsbrunner H, Guibas L. Topologically sweeping an arrangement. Comp. Syst. Sci., 1989,38: 165~194

102. Edelsbrunner H, Seidel R. Voronoi diagrams and arrangements. Disc. Comp. Geom., 1986,(1): 25~44

103. Edelsbrunner H, Seidel R, Sharir M. On the zone theorem for hyperplane arrangements. SIAM J. Comp., 1993,22(2):418~429

104. Edelsbrunner H, Welzl E. Halfplanar range search in linear space and $O(n^{0.695})$ query time. Information Processing Letters, 1986,23:289~293

105. Elzing J, Hearn D W. Geometrical solutions for some minimax location problems. Transportation Science, 1971,6:379~394

106. Even S. Graph algorithms. Potomac: Computer Science Press, MD, 1979

107. Fary I. On straight-line representation of planar graphs. Acta Sci. Math. Szeged., 1948,(11): 229~233

108. Forrest A R. Computational geometry, Proc. Royal Society London, 1971,321 Series 4,187~195

109. Fortune S. Voronoi diagrams and Delaunay triangulations. In: Du D Z(ed). Computing in Euclidean geometry, World Scientific Publishing Co. Pte. Ltd, 1995

110. Fortune S. A sweepline algorithm for Voronoi diagrams. Algorithmica, 1987,(2):153~174

111. Garey M R, Johnson D S,Preparata F P, Tarjan R E. Triangulating a simple polygon. Inform. Process. Lett., 1978,(7):175~180

112. Gass S I. Linear programming. New York: McGraw-Hill, 1969

113. Ghosh S K, Maheshwari A. An optimal parallel algorithm for determining the intersection type of two star-shaped polygons. In: Proc. 3rd Canad. Conf. Comput. Geom., Vancouver, 1991,2~6

114. Ghosh S K, Mount D M. An output sensitive algorithm for Computing visibility graphs. Proc. of the 28th Ann. Symp. on Foundations of Computer Science, 1987,11~19

115. Ghosh S K, Mount D M. An output-sensitive algorithm for computing visibility graphs. SIAM J. Comput., 1991,20(5):888~910

116. Gilbert P N. New results on planar triangulations. Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979

117. Gilbert E N, Pollak H O. Steiner minimal trees. SIAM J. Appl. Math., 1968,(16):1～29

118. Goodrich M T. Geometric partitioning made easier, even in parallel. In: Proc. 9th Annu. ACM Sympos. Comput. Geom., 1993,73～82

119. Goodrich M T. Fixed-dimensional parallel linear programming via relative epsilon-approximations. In: Proc. 7th ACM-SIAM Sympos. Discrete algorithms, 1996,132～141

120. Goodrich M T. Parallel algorithms in geometry. In: Goodman J E, O'Rourke J(eds). Handbook of Discrete and Computational Geometry. New York: CRC Press LLC, 1997

121. Goodrich M T. Planar separators and parallel polygon triangulation. Proc. 24th Ann. ACM Symp. Theory Comput. 1992,507～516

122. Goodrich M T, O'Dúnlaing C, Yap C. Computing the Voronoi diagram of a set of line segments in parallel. Algorithmica, 1993,9:128～141

123. Goodrich M T, Shauck S, Guha S. Parallel methods for visibility and shortest path problems in simple polygons. Algorithmica, 1992,8:461～486

124. Graham R L. An efficient algorithm for determining the convex hull of a finite planar set. Info. Proc. Lett., 1972,1:132～133

125. Green P J, Silverman B W. Constructing the convex hull of a set of points in the plane. Computer Journal, 1979,22:262～266

126. Greene D H. The decomposition of polygons in convex parts, In: Preparata F P(ed). Advances in Computing Research, JAI Press, 1983,235～259

127. Grunbaum B. Convex polytopes. New York: John Wiley and Sons, 1967

128. Guibas L J, Hershberger J. Optimal shortest path queries in a simple polygon. Proc. of the 3th Ann. Symp. on Computational Geometry, 1987,50～63

129. Guibas L J, Stolfi J. Ruler, compass and computer: The design and analysis of geometric algorithms. In: Earnshaw R A(ed). Theoretical Foundations of Computer Graphics and CAD in NATO ASI F40, Springer-Verlag, 1988,111～165

130. Guibas L J, Stolfi J. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Trans. Graphics, 1985,4(2):74～123

131. Guibas L J, Yao F F. On translating a set of rectangles. In: Preparata F P(ed) Computational Geometry, in Advances in Computing Research, Vol. 1, London: JAI Press, England, 1983,61～77

132. Hanan M. On Steiner's problem with rectilinear distance. J. SIAM Appl. Math. 1966,(14):255～265

133. Hershberger J. Optimal parallel algorithms for triangulated simple polygons. In: Proc. 8th Annu. ACM Sympos. Comput. Geom., 1992,33～42

134. Hershberger J, Suri S. A pedestrian approach to ray shooting: Shoot a ray, take a walk. In: Proc. 4th ACM-SIAM Sympos. Discrete Algorithms. 1993,54～63

135. Herrel S, Mehlhorn K. Fast triangulation of simple polygons. In: Proc. 4th Internat. Conf. Found. Comput. Theory, In: Lecture Notes in Computer Science, Vol. 158. 1983,207～218

136. Hopcroft J, Joseph D, Whitesides S. On the movement of robot arms in 2-dimensional bounded regions. SIAM J. Comp., 1985,14(2):315～333

137. Hsu F R, Chang R C, Lee R C T. Parallel algorithms for computing the closest visible vertex pair between two polygons. Int'l J. Comput. Geom. Appl., 1992,(2):135～162

138. Hwang F K. On Steiner minimal trees with rectilinear distance. SIAM J. Appl. Math., 1976, (30): 104～114

139. Hwang F K. A linear time algorithm for full Steiner trees. Operations Research Letters 5, 1986, 235～237

140. Hwang F K, Song G D, Ting G Y, Du D Z. A decomposition theorem on Euclidean Steiner minimal trees. Discrete and Computational Geometry. 1988,(3):367～382

141. Hwang F K, Weng J F, Du D Z. A class of full Steiner minimal trees. Discrete Mathematics, 1983,(45):107～112

142. Kallay M. Convex hall algorithms in higher dimensions. Unpublished manuscript, Dept. of Mathematics, Univ. of Oklahoma, Norman, Oklahoma, 1981

143. Kantabutra V. Motion of a short-linked robot arm in a square. Disc. Comp. Geom., 1992,(7): 69～76

144. Kantabutra V, Kosaraju S R. New algorithms for multilink robot arms, J. Comp. Syst. Sci., 1986,32(1):136～153

145. Kaul A, O'Connor M A, Srinivasan V. Computing Minkowski sums of regular polygons. Proc. 3rd Canad. Conf. Comput. Geom., 1991, 47～77

146. Kedem K, Sharir M. An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space. Disc. Comp. Geom., 1990,(5):43～75

147. Keil J M. Decomposing a polygon into simpler components. SIAM J. Comp., 1985,(14):799～817

148. Keil J M. Sack J R. Minimum decomposition of polygonal objects. In: Toussaint G T(ed). Computational Geometry, Amsterdam: North-Holland, 1985

149. Kirkpatrick D G. Optimal search in planar subdivisions. SIAM J. Comput., 1983,12(1):28～35

150. Kirkpatrick D G. Klawe M M, Tarjan R E. Polygon triangulation in $O(n \text{ loglog } n)$ time with simple data structures. Disc. Comput. Geom., 1992,(7):329～346

151. Kirkpatrick D G, Seidel R. The ultimate planar convex hull algorithm? SIAM J. Comput., 1986, (15):287～299

152. Komlos J, Shing M T. Probabilistic partitioning algorithms for the rectilinear Steiner problem. Networks, 1985,(15):413～423

153. Korhonen P. An algorithm for transforming a spanning tree into a Steiner tree. Proc. of 9th Mathematical programming Symp., 1979,(2):349～357

154. Kutcher J. Coordinated motion planning of planar linkages. Ph. D. Thesis, Johns Hopkins University, 1992

155. Lawler E L, Lenstra J K, Rinnooy Kan A H G, Shmoys D. The traveling salesman problem. John Wiley and Sons. 1985

156. Lee D T. Two dimensional Voronoi diagram in the Lpmetric, J. ACM, 1980,27:604～618

157. Lee D T, Preparata F P. The all nearest neighbor problem for convex polygons. Info. Proc. Lett., 1978,7:189～192

158. Lee D T, Preparata F P. An optimal algorithm for finding the kernel of a polygon. Journal of the ACM, 1979,26:415～421

159. Lee D T, Preparata F P. Euclidean shortest paths in the presence of rectilinear barriers. Networks, 1984,14:393～410

160. Lee D T, Wong C K. Voronoi diagrams in L_1 (L_∞) metrics with 2-dimensional storage applications. SIAM J. Comput. 9, 1980:200~211

161. Lee J H, Bose N K, Hwang F K. Use of Steiner's problem in suboptimal routing in rectilinear metric. IEEE Trans. on circuits and systems, 1976,CAS-23:470~476

162. Leven D, Sharir M. An efficient and simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. J. Algor., 1987,(8):192~215

163. Liestman A. Construction of Steiner trees with obstacles in the plane. M. Sc. Thesis, Dept. of Computer Science, University of Illinois, Urbana, Illinois, 1978

164. Lipski W Jr, Preparata F P. Segments, rectangles, contours. Journal of Algorithms, 1981,2:63~76

165. Matoušek J. Efficient partition trees. Proc. 7th ACM Comput. Geom., 1991,1~9

166. Matoušek J. Derandomization in computational geometry. In: Sack J R, Urrutia J(eds). Hand book on Compulational Geometry. Amsterdam: North Holland, to appea. Earlier version in J. Algorithms, 1995,(20):545~580

167. Matoušek J. Schwarzkopf O, Snoeyink J. Noncanonical randomized incremental construction. Manuscript, 1996

168. McKenna M. Worst-case optimal hidden-surface removal. ACM Trans. Graph., 1987,6:19~28

169. McKenna M. Applications of arrangements to geometric problems in higher dimensions, Ph. D. Thesis, Johns Hopkins University, 1989

170. McKenna M, Seidel R. Finding the optimal shadows of a convex polytope. Proc. 1st Annu. ACM Symp. Comp. Geom., 1985,24~28

171. Megiddo N. Linear time algorithm for linear programming in R^3 and related problems. SIAM J. Comput., 1983,12(4):759~776

172. Mehlhorn K. Data structures and algorithms 3: Multidimensional searching and computational geometry. Springer-Verlag, Heidelberg, Germany, 1984

173. Melachrinoudis E. The maximin single facility location problem using a Euclidean metric. Ph. D. Thesis. Dept. of Industrial Engineering and Operations Research, University of Massachusetts. Amherst, MA, 1980

174. Melachrinoudis E, Cullinane T P. A heuristic approach to the single facility maximin location problem. Int. J. Prod. Res., 1985,(23):523~532

175. Melachrinoudis E, Cullinane T P. Locating an undesireable facility within a geographic region using the MAXIMIN criterion. J. Reg. Sci., 1985,(25):115~127

176. Melachrinoudis E, Cullinane T P. Locating an obnoxious facility within a polygonal region. Annals of Operations Research, 1986,6:137~145

177. Melzak Z A. On the problem of Steiner. Canad. Math. Bull., 1961,4:143~148

178. Mitchell J S B, Papapdimitriou C H. The weighted region problem. Proc. of the 3rd Ann. Symp. on Computational Geometry, 1987,30~38

179. Monma C, Paterson M, Suri s, Yao F. Computing Euclidean maximum spanning trees. Algorithmica, 1990,(5):407~419

180. Motwani R, Raghavan P. Randomized algorithms. Cambridge University Press, 1995

181. Mulmuley K. A fast planar partition algorithm, I. Proc. 5th ACM Comput. Geom., 1989,33~43

182. Mulmuley K. Computational Geometry: An Introduction through randomized algorithms. Englewood Cliffs: Prentice Hall, 1994

183. Mulmuley K, Schwarzkopt O. Randomized algorithms. In: Goodman J E, O'Rourke J(eds). Handbook of Discrete and Computational Geometry, New York: CRC Press LLC, 1997

184. Ó'Dúnlaing C, Yap C K. A "retraction" method for planning the motion of a disk. J. Algor., 1985,(6):104～111

185. Okabe A, Boots B, Sugihara K. Spatial tessellations: Concepts and applications of Voronoi diagrams. Wiley, 1992

186. O'Rourke J. Computational geometry in C. Cambridge Univ. Press, 1994

187. O'Rourke J. Art gallery theorems and algorithms. New York: Oxford University Press, 1987

188. O'Rourke J. Computational geometry. Annu. Rev. Comp. Sci., 1988,(3):389～411

189. Papadimitrion C H. An algorithm for shortest path motion in three dimensions. Information Processing Letters, 1985,20:259～263

190. Plantinga H, Dyer C R. Visibility, occlusion, and the aspect graph. Int. J. Comput. Vis., 1990, 5(2):137～160

191. Preparata F P. An optimal real time algorithm for planar convex hulls. Comm. ACM, 1979,22: 402～405

192. Preparata F P, Hong S J. Convex hulls of finite sets of points in two and three dimensions. Comm. ACM, 1977,2(20):87～93

193. Preparata F P, Shamos M I. Computational geometry: An introduction. New York: Springer-Verlag, 1985

194. Preparata F P, Shamos M I. Computational geometry: An introduction. New York: Springer-Verlag, 1988

195. Provan J S. Convexity and Steiner tree problem. Networks, 1988,(18):55～72

196. Provan J S. An approximation scheme for finding Steiner trees with obstacles. SIAM J. Comput., 1988,(17):920～934

197. Reif J H, Sen S. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. SIAM J. Comput., 1992,(21):466～485

198. Reif J, Storer J A. Motion planning in the presence of moving obstacles. Proc of the 26th Symp. on the Foundations of Computer Science, 1985,144～154

199. Reif J, Storer J A. Shortest paths in Euclidean space with polyhedral obstacles. Technical Report CS-85-121, Computer Science Department, Brandeis University, Waltham, MA, 1985

200. Richards D. Fast heuristic algorithms for rectilinear Steiner trees, Algorithmica, 1989,(4):191～207

201. Rohnert H. Shortest paths in the plane with convex polygonal obstacles. Information Processing Letters, 1986,23:71～76

202. Rohnert H. Time and space efficient algorithms for shortest paths between convex polygons. Information Processing Letters, 1988,27:175～179

203. Schwarzkopf O, Sharir M. Vertical decomposition of a single cell in a three-dimensional arrangement of surfaces and its applications. Proc. 12th Annu. ACM Sympos. Comput. Geom., 1996,20～29

204. Schwartz J T, Sharir M. On the "piano movers" problem Ⅰ: the case of a two-dimensional rigid

polygonal body moving amidst polygonal barriers. Commun. Pure. Appl. Math., 1983a,(36): 345～398

205. Schwartz J T, Sharir M. On the "piano movers" problem Ⅱ: general techniques for computing topological properties of real algebraic manifolds. Adv. Appl. Math., 1983b,(4):298～351

206. Schwartz J T, Sharir M. On the "piano movers" problem Ⅲ: coordinating the motion of several independent bodies: the special case of circular bodies moving amidst polygonal barriers. Int'l. J. Rob. Res., 1983c, 2(3):46～75

207. Schwartz J T, Sharir M. On the "piano movers" problem Ⅴ: the case of a rod moving in three-dimensional space amidst polyhedral obstacles. Commun. Pure Appl. Math., 1984

208. Schwartz J T, Sharir M. A survey of motion planning and related geometric algorithms. Artif. Intell., 1988,(37):157～169

209. Schwartz J T, Sharir M. Algorithmic motion planning in robotics. In: Van Leeuwen J(ed). Algorithms and Complexity, Handbook of Theoretical Computer Science, Vol. A, Amsterdam: Elsevier, 1990,391～430

210. Seidel R. Constructing higher-dimensional convex hull algorithms at logarithmic cost per face. Proc. 18th Ann. Symp. Theory. Comp., 1986,404～413

211. Seidel R. Backwards analysis of randomized geometric algorithms. In: Pach J(ed). New Trends in Discrete and Computational Geometry, Volume 10 of Algorithms Combin., Berlin: Springer-Verlag, 1993,67～68

212. Seidel R. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput. Geom. Theory Appl., 1991,(1):51～64

213. Shamos M I. Problems in computational geometry. Unpublished manuscript, 1975a

214. Shamos M I. Computational geometry. Ph. D. Thesis, Dept. of Comput. Sci., Yale Univ., 1978

215. Shamos M I, Hoey D. Closest-point problems. Proc. 16th IEEE Ann. Symp. on the Foundations of Computer Science,1975,151～162

216. Sharir M. On shortest paths amidst convex polyhedra. SIAM J. Comput., 1987,(16):561～572

217. Sharir M, Schorr A. On shortest paths in polyhedral spaces. SIAM J. Comput. 1986,(15):193～215

218. MacGregor Smith J. Steiner minimal trees with obstacles. Paper presented at the ORSA/TIMS Meeting Detroit, Michigan, 1982

219. MacGregor Smith J, Lee D T, Liebman J S. An $O(n \log n)$ heuristic algorithm for the rectilinear Steiner minimal tree problem. Engineering Optimization, 1980,(4):179～192

220. MacGregor Smith J, Lee D T, Liebman J S. An $O(n \log n)$ heuristic for Steiner minimal tree problems on the Euclidean metric. Networks, 1981,(11):23～29

221. MacGregor Smith J, Liebman J S. Steiner trees, Steiner circuits, and the interference problem in building design. Engineering Design, 1979,(4):15～36

222. MacGregor Smith J, Weiss R, Patel M. An $O(N^2)$ heuristic for the Steiner minimal tree problem in E^3, Paper presented at the ORSA/TIMS meeting, Chicago Illinois., May 1993, in review

223. MacGregor Smith J, Winter P. Computational geometry and topological network design. In: Du D Z, Hwang F(eds). Computing in Euclidean geometry, Singapore: World Scientific Publish Co.

Pte. Ltd., 1995, Volume 4, 351～451

224. Smith W D. How to find Steiner minimal trees in Euclidean d-space. Algorithmica, 1992, (7): 137～177

225. Smith W D, MacGregor Smith J. On the Steiner ratio in 3-space. Paper accepted in Journal of Combinatorial Theory, Series A, 1994

226. Steele J M, Yao A C. Lower bounds for algebraic decision trees. J. Algorithms, 1982, 3: 1～8

227. Stewart Jr. J. A computationally efficient heuristic for the traveling salesman problem. Proc. 13th Ann. Meeting of S. E. TIMS, 1977, 75～85

228. Tagansky B. A new technique for analyzing substructures in arrangements. Proc. 11th Annu. ACM Sympos. Comput. Geom., 1995, 200～210

229. Tamassia R, Vitter J S. Parallel transitive closure and point location in planar structures. SIAM J. Comput., 1991, (20): 708～725

230. Tarjan R E, Van Wyk C J. An $O(n \log\log n)$-time algorithm for triangulating a simple polygon. SIAM J. Comput., 1988, (17): 143～178

231. Tokuyama T, Orthogonal queries in segments and triangles. In: Du D Z, Zhang X S (eds). Algorithms and Computation, 5th Inter. Symp. ISAAC '94, Berlin: Springer-Verlag, Heidelberg, 1994

232. Trietsch D, Hwang F K. An improved algorithm for Steiner trees. SIAM J. Appl. Math., 1990, (50): 244～263

233. Vaidya P M. Minimum spanning trees in k-dimensional space. SIAM J. Comput., 1988, (17): 572～582

234. Watanabe T, Sugiyama Y. A new routing algorithm and its hardware implementation. Proc. of the 23th ACM/IEEE Design Automation Conf., 1986, 574～580

235. Welzl E. Constructing the visibility graph for n line segments in $O(n^2)$ time. Information Processing Letters, 1985, 20: 167～171

236. Whitesides S. Algorithmic issues in the geometry of planar linkages. Aust. Comput. J., 1991, 24 (2): 42～51

237. Widmayer P. On graphs preserving rectilinear shortest paths in the presence of obstacles. Annals of Operations Research, 1991, (33): 557～575

238. Winter P. An algorithm for the Steiner problem in the Euclidean plane. Networks, 1985, 15: 323～345

239. Winter P, MacGregor Smith J. Steiner minimal trees with obstacles: 3 and 4 point case. Paper presented at the NATO/ARW on Topological Network Design Schaeffergarden, Denmark, 1989

240. Winte P, MacGregor Smith J. Path-distance heuristics for the Steiner problem in undirected networks. Algorithmica, 1992, 7(2/3): 309～327

241. Winter P, MacGregor Smith J. Steiner minimal trees for three points with one convex ploygonal obstacle. Annals of Operations Research, 1991, (33): 577～599

242. Yang Y Y, Wing O. Optimal and suboptimal solution algorithms for the wiring problem. Proc. of the IEEE Int. Symp. on Circuit. Theory, 1972, 154～158

243. Yao A C. Space-time tradeoff for answering range queries. Proc. 14th ACM STOC, 1982, 128～136

244. Yao A C. On constructing minimum spanning trees in k-dimensional space and related problems.

SIAM J. Comput., 1982,(11):721～736

245. 周培德、周忠平,张欢. 寻求中国货郎担问题最短回路的多项式时间算法. 北京理工大学学报,待发表

246. Deering M. Geometry compression. In: Computer Graphics Proceedings, Annual Conference Series. New York: ACM SIGGRAPH, ACM Press,1995,13～20

247. Stefan Gumhold, Wolfgany Stroβer. Real time compression of triangle mesh connectivity. In: Computer Graphics Proceedings, Annual Conference Series. New York: ACM SIGGRAPH, ACM Press, 1998,133～140

248. Taubin G, Rossignac J. Geometry compression through topological surgery. ACM Trans. on Graphics, 1998, 17(2): 84～115

